Sistemas Informáticos
Curso 2004-2005

Desarrollo de Servicios Web para la gestión de objetos virtuales en entornos de e-learning

Laura Gómez Pérez
Beatriz Molina Sánchez
Álvaro Rodrigo Yuste

Dirigido por:
José Luis Sierra Rodríguez
Dpto. Sistemas Informáticos y Programación

Facultad de Informática
Universidad Complutense de Madrid

INDICE

AGRADECIMIENTOS

Nuestro más sincero agradecimiento a nuestro director de proyecto por su ayuda y apoyo constante y al grupo de sistemas informáticos que ha trabajado con el MIGS durante este curso por su colaboración.

RESUMEN

El proyecto *'Desarrollo de Servicios Web para la gestión de objetos virtuales en entornos de e-learning'* se ha desarrollado en la asignatura de Sistemas Informáticos del curso 2004/2005.

Tiene como objetivo el desarrollo de un Servicio Web que actúa como una interfaz programática sobre el Museo Virtual de Informática García Santesmases (MIGS) de la Facultad de Informática y el Museo Virtual CHASQUI de la Facultad de Geografía e Historia de la Universidad Complutense de Madrid. Para probar el funcionamiento se ha desarrollado una herramienta que permite gestionar varios museos virtuales.

El sistema publica las funcionalidades de ambos museos como una serie de operaciones, haciendo que el acceso a estos dos museos confluya en un único Servicio Web. Por la modularidad del sistema implementado, se pueden realizar futuras ampliaciones. Además el Servicio Web facilita el uso de distintos mecanismos de acceso (como por ejemplo los basados en dispositivos móviles) y el uso de distintas herramientas de autoría, de tal modo que puedan conectarse simultáneamente a diferentes repositorios usando el interfaz basado en Servicios Web.

ABSTRACT

The project 'Desarrollo de servicios web para la gestión de objetos virtuales en entornos de e-learning' has been developed in the subject of Sistemas Informáticos during the course 2004/2005.

The goal has been the construction of a Web Service that is used as a programmatic interface over the virtual museums MIGS and CHASQUI. To test how it works, we also have developed a web based tool to manage several virtual museums.

The system makes to converge the access to both museums by publishing its functionality as a list of operations. Due to the modularity of the system, it is possible to work over future extensions. Furthermore, the Web Service makes easy the use of different mechanisms to access its functionality (for example, those based on mobile devices) and the use of different tools of edition, so it is possible the connection of several repositories simultaneously using interfaces based on Web Services.

1. INTRODUCCIÓN

Este documento presenta el trabajo realizado para el proyecto *'Desarrollo de Servicios Web para la Gestión de Objetos virtuales en Entornos de e-learning'*.

El proyecto consiste en la creación de una interfaz programática mediante un Servicio Web para los museo virtuales de la Universidad Complutense de Madrid MIGS (Museo de Informática García Santesmases) de la Facultad de Informática y CHASQUI de la Facultad de Geografía e Historia.

1.1. ESTADO DEL ARTE

Para entender totalmente el significado de este proyecto, es necesario describir el marco en que está inmerso.

El proyecto implementa un Servicio Web que añade interfaces programáticas a las funcionalidades de los museos virtuales MIGS y CHASQUI pertenecientes a la Universidad Complutense de Madrid (UCM).

1.1.1. MUSEOS Y OBJETOS VIRTUALES

Un *museo virtual* es una interfaz accesible a través de cualquier computador que tenga acceso a la red y disponga de un navegador web, permitiendo consultar toda la información relativa a un museo real y, especialmente, la relativa a los objetos expuestos en el mismo.

El concepto de *objeto virtual* es un modelo de organización de la información digital de objetos en el mundo real que facilita el uso educacional de dicha información. Dicho concepto ha surgido durante el desarrollo de un par de museos virtuales accesibles como aplicaciones *web*, y se basa en el concepto de *objeto de aprendizaje* utilizado en el ámbito de las tecnologías *e-learning* [1]. Los objetos virtuales constan de:

- Un conjunto de *datos* que representan todas aquellas características del objeto potencialmente útiles para su estudio científico y su uso didáctico.

- Un conjunto de *meta-datos* que describen y clasifican el objeto desde el punto de vista de su utilidad docente.

- Un conjunto de *recursos*. Los recursos pueden ser a su vez de tres tipos:

    - recursos *propios* (material multimedia dado en términos de archivos digitales de texto, imágenes, audio o video),

    - recursos *pertenecientes a otros objetos* (recursos que tienen algún tipo de relación con el objeto pero que son propiedad de otros objetos virtuales); y

- recursos que son *otros objetos virtuales*. Mediante esta relación de inclusión es posible crear jerarquías sencillas de objetos virtuales.

En el caso de un museo virtual, los objetos virtuales representan los objetos físicos expuestos en el museo así como aquellos que por distintos motivos no son accesibles al público.

Por tanto, los museos virtuales permiten un acceso remoto y en ocasiones más amplio que un museo real.

1.1.2. ORIGEN DEL MUSEO VIRTUAL CHASQUI

Un problema que afecta a las grandes universidades, y en particular a las más antiguas, como es el caso de la Universidad Complutense de Madrid (UCM), es el de carecer de medios que les permitan explotar, en todo su potencial investigador y docente, su patrimonio histórico-cultural, acumulado a lo largo de los años en pequeños museos, o depósitos, que solo disponen de un acceso *real* muy restringido.

Como respuesta a esta limitación, se inició el proyecto Chasqui, por parte del departamento de Sistemas Informáticos y Programación de la Facultad de Informática en colaboración con el Departamento de Historia de América II de la Facultad de Geografía e Historia. El objetivo era definir un *museo virtual*, que permitiera *publicar* el material arqueológico depositado en el museo y en el laboratorio de este último departamento, facilitando de esta forma su utilización con fines docentes y de investigación.

Como resultado de este primer proyecto surgió el repositorio Chasqui, que ofrece una interfaz web para consultar los *objetos virtuales*, que representaban los objetos físicos disponibles en las dependencias de la facultad, sus recursos digitales asociados, y las relaciones existentes entre ellos [2].

Se puede decir que la idea en que se basa el sistema Chasqui ha sido un éxito pues la información contenida en su repositorio está siendo muy utilizada tanto por investigadores como por profesores y alumnos. Más departamentos en la UCM han mostrado su interés en ofrecer sus recursos de una manera similar, para facilitar la divulgación, la docencia y la investigación.

1.1.3. ORIGEN DEL MUSEO VIRTUAL MIGS

Físicamente, el Museo de Informática García Santesmases está localizado en la Facultad de Informática y exhibe varias máquinas desarrolladas en la UCM entre 1950 y 1975, varios ordenadores comerciales que han sido utilizados en el centro de cálculo de la universidad desde 1968, y otros equipamientos donados por la universidad, otras entidades y otras donaciones a título personal. El museo también tiene diferentes tipos de material documental tal como manuales, fotos y registros de investigación de los pioneros en informática en la UCM.

La mayoría de estos objetos tienen un gran valor pedagógico porque representan una perspectiva íntegra de la evolución de la informática en España. Sin embargo, el acceso educativo al material exhibido en el museo es difícil por las características de este museo. En principio, los equipos están localizados dentro de urnas de cristal con tarjetas informativas que describen algunas características de estos equipos. Además el museo no tiene personal para proporcionar información sobre dichos equipos. Por último, no hay folletos sobre los fondos y solamente una pequeña parte del material documental que el museo tiene está expuesto. En este contexto, e inspirándose en el proyecto Chasqui, se propuso un proyecto para la virtualización de este museo que además permitiera acceder a ciertos objetos que por diversas razones no están expuestos, tales como documentos y fotografías [3].

El proyecto de este museo virtual ha intentado solucionar muchos de los problemas que tiene el Chasqui. Se intenta facilitar la gestión y mantenimiento mejorando el diseño de su base de datos para que toda la información y el material educativo sea consistente. Otra de las mejoras ha sido permitir que el contenido del museo sea accesible y exportable a otros entornos.

1.1.4. TECNOLOGÍAS EN QUE SE BASAN AMBOS MUSEOS

En ambos museos el repositorio principal de los *objetos virtuales* se ejecuta en una máquina (actualmente un servidor Apple), donde se aloja el servidor de Bases de Datos MySQL y el motor de PHP que proporciona la interfaz web de usuario final.

1.1.4.1. PHP

PHP (Hypertext Preprocessor) [4] es un lenguaje del lado del servidor (esto significa que PHP funciona en el servidor remoto que procesa la página web antes de que sea abierta por el navegador del usuario) especialmente creado para el desarrollo de páginas web dinámicas. Puede ser incluido con facilidad dentro del código HTML, y permite una serie de funcionalidades tan extraordinarias que han hecho que se haya convertido en el favorito de millones de programadores en todo el mundo.

Combinado con la base de datos MySQL, es el lenguaje estándar a la hora de crear sitios de comercio electrónico o páginas web dinámicas.

Las páginas web dinámicas añaden la ventaja sobre las estáticas de que cuando se está procesando la petición del visitante, dependiendo de factores como las preferencias del usuario, su país de procedencia o su idioma natal, solicitará unos contenidos específicos a la base de datos, con lo que la página mostrada será absolutamente personalizada y adaptada a las características del visitante. Asimismo, las modificaciones y actualizaciones en la página web dinámica pueden ser realizadas de manera sumamente sencilla, sin necesidad de saber HTML ni utilizar ningún gestor de FTP, por lo que cualquier persona autorizada podrá realizar cambios. Otras tecnologías para páginas dinámicas son ASP (Active Server Pages) [5] y JSP (Java Server Page) [6].

Entre las características fundamentales de PHP están:

- *Gratuidad*. Al tratarse de software libre, puede descargarse y utilizarse en cualquier aplicación, personal o profesional, de manera completamente libre.

- *Gran popularidad*. Existe una gran comunidad de desarrolladores y programadores que continuamente implementan mejoras en su código, y que en muchos casos estarán dispuestos a echar una mano cuando surja algún problema. Actualmente son casi diez los millones de páginas web desarrolladas con PHP.

- *Sencilla integración con múltiples bases de datos*. Esencial para una página web verdaderamente dinámica, es una correcta integración con bases de datos. Aunque MySQL es la base de datos que mejor trabaja con PHP, puede conectarse también a PostgreSQL, Oracle, dbm, filePro, interbasem o cualquier otra base de datos compatible con ODBC (Open Database Connectivity Standard) [7].

- *Versatilidad*. PHP puede usarse con la mayoría de sistemas operativos, ya sean basados en UNIX o en Windows.

- *Gran número de funciones predefinidas*. A diferencia de otros lenguajes de programación, PHP fue diseñado especialmente para el desarrollo de páginas web dinámicas. Por ello, está dotado de un gran número de funciones que simplifican tareas habituales como descargar documentos, enviar correos, trabajar con cookies y sesiones, etc.

1.1.4.2. MYSQL

MySQL [8] es una de las bases de datos más populares desarrolladas bajo la filosofía de código abierto.

La desarrolla y mantiene la empresa MySQL AB pero puede utilizarse gratuitamente y su código fuente está disponible.

Inicialmente, MySQL carecía de elementos considerados esenciales en las bases de datos relacionales, tales como integridad referencial y transacciones. A pesar de ello, atrajo a los desarrolladores de páginas web con contenido dinámico, justamente por su simplicidad. Aquellos elementos omitidos fueron llenados por la vía de las aplicaciones que la utilizan.

Poco a poco los elementos omitidos en MySQL están siendo incorporados tanto por desarrolladores internos, como por desarrolladores de software libre. Entre las características disponibles en las últimas versiones se puede destacar:

- Amplio subconjunto del lenguaje SQL (Structured Query Language). Así mismo incluye algunas extensiones de dicho lenguaje.
- Diferentes opciones de almacenamiento según si se desea velocidad en las operaciones o el mayor número de operaciones disponibles.
- Transacciones y claves foráneas.
- Conectividad segura.
- Replicación.
- Búsqueda e indexación de campos de texto.

Según las cifras del fabricante, existen más de seis millones de copias de MySQL funcionando en la actualidad, lo que supera el número de instalaciones de cualquier otra herramienta de bases de datos.

La licencia GPL (General Public License) [9] de MySQL obliga a distribuir cualquier producto derivado (aplicación) bajo esta misma licencia. Si un desarrollador desea incorporar MySQL en su producto pero no desea distribuirlo bajo licencia GPL, puede adquirir la licencia comercial de MySQL que le permite hacer justamente eso.

MySQL es, sin duda, la base de datos más popular y utilizada a la hora de desarrollar páginas web dinámicas y sitios de comercio electrónico. Se suele trabajar en combinación con PHP, y comparte con éste algunas de las características que lo convierten en una elección segura. Entre ellas:

- *Popularidad*. Son innumerables las páginas donde encontrar información, y las listas de correo de donde poder obtener ayuda.

- *Rapidez*. La velocidad de proceso de MySQL es legendaria.

- *Versatilidad*. Trabaja tanto con sistemas operativos basados en Unix como en el sistema operativo Windows.

- *Sencillez de manejo*. Al utilizar el lenguaje estándar SQL, la rampa de entrada para un desarrollador con conocimientos generales de bases de datos es pequeña.

1.2. NECESIDADES Y OBJETIVOS

Como se ha expuesto en la sección anterior, los museos virtuales solucionan bastantes problemas que tienen los museos reales además de servir como apoyo a la educación. Pero también el software orientado a la educación presenta problemas que este proyecto trata de solucionar.

Desde sus pasos iniciales en los años 60 hasta nuestros días, el principal problema de todo software para la educación ha sido la falta de independencia de la plataforma y su reducida reusabilidad y mantenibilidad. Esta situación ha provocado un tipo de software con un ciclo de vida muy corto. Algunos autores creen que esta es la razón de la limitada adopción de este tipo de aplicaciones [10].

Las aplicaciones para el manejo de los museos virtuales anteriormente descritos (MIGS y CHASQUI) no son reusables directamente en otros contextos porque aunque tienen principios similares, ambas han sido desarrolladas para solucionar los problemas particulares que aparecen en cada museo. Para superar estas dificultades se ha propuesto el uso de Servicios Web.

Originariamente se parte de la estructura mostrada en la figura 1. En esta arquitectura se tiene un repositorio de objetos virtuales a los que se accede tanto para su consulta como para su administración a través de la interfaz web desarrollada en PHP para tal efecto.

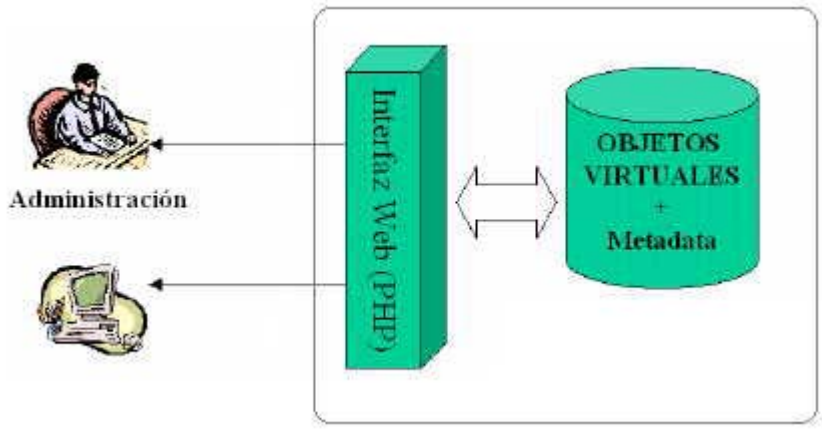

Figura 1. Arquitectura actual de MIGS y CHASQUI

Sin embargo esta arquitectura presenta como principal problema la interoperabilidad.

Actualmente es deseable en los accesos a repositorios de objetos virtuales tener aplicaciones que trabajen de forma autónoma sin la supervisión de un humano. Para ambos museos, estas aplicaciones deberían acceder a la interfaz web para realizar desde ella operaciones tales como consultas, modificación de objetos y otras más en el caso de herramientas de autoría.

El acceso a la interfaz es fácil de programar pero presenta el inconveniente de que dado el carácter presentacional de HTML, un simple cambio en el código de la interfaz provocaría un fallo en las aplicaciones que realizan el acceso.

Las tecnologías de Servicios Web han aparecido como una solución para la interoperabilidad de aplicaciones en Internet.

Como objetivo de los sistemas MIGS y CHASQUI, surgió por tanto el desarrollo de un Servicio Web que mejorase ambas arquitecturas. El objetivo final es alcanzar la arquitectura expuesta en la figura 2. En esta arquitectura, se mantiene la interfaz web de la anterior arquitectura pero se añade el acceso al repositorio mediante Servicio Web.

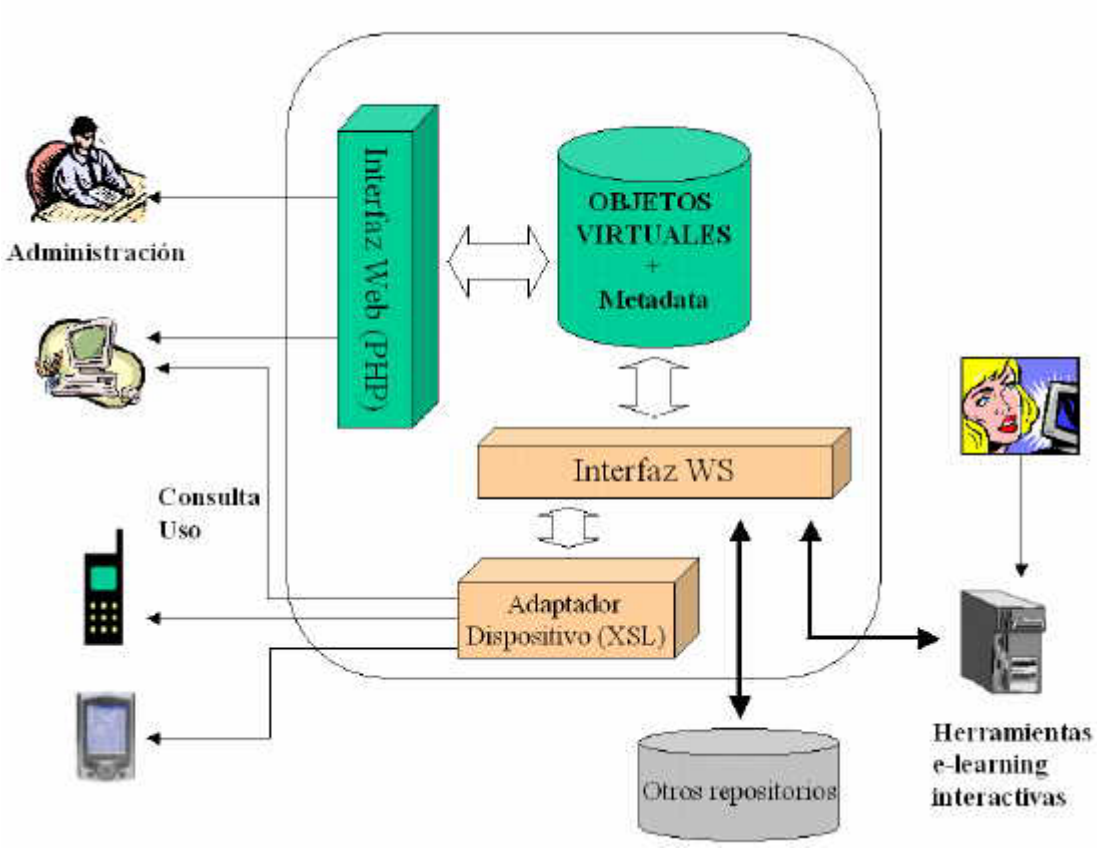

Figura 2. Arquitectura final deseada para MIGS y CHASQUI.

Esta arquitectura permite realizar funciones, tanto de consulta como de actualización del repositorio, sin necesidad de utilizar la interfaz de usuario final (interfaz web).

La interfaz basada en Servicios Web facilita el uso de distintos mecanismos de acceso (como por ejemplo los basados en dispositivos móviles) y también el uso de distintas herramientas de autoría que puedan conectarse simultáneamente a diferentes repositorios usando la interfaz basada en Servicios Web. Además se pretende que el uso del Servicio Web facilite la reutilización de objetos virtuales en entornos heterogéneos de e-learning.

La interfaz basada en Servicios Web permitirá usar objetos de aprendizaje del museo virtual y otros repositorios provistos de una interfaz similar. Esta clase de interfaces también mejorará la interoperabilidad entre repositorios de contenidos. Además, la arquitectura propuesta explota la simplicidad para desarrollar clientes basados en estándares en un contexto de Servicios Web ya que una de las grandes ventajas de esta interfaz es su independencia del lenguaje de programación y la plataforma, además de la facilidad para crear clientes que se conecten a dicha interfaz al existir herramientas que automatizan el proceso. Además, si fuese necesario cambiar la implementación interna del Servicio Web pero manteniendo la interfaz de acceso, las aplicaciones que antes accedían podrían seguir haciéndolo normalmente.

Otra ventaja es que la nueva arquitectura permitirá el uso de objetos virtuales en medios heterogéneos por medio de definiciones de clientes orientados a terminal (teléfonos móviles, PDAs) usando hojas de estilo XSL (eXtensible Stylesheet Language) [11], para transformar la salida genérica XML en lenguajes de marcado específicos, como WML (Wireless Markup Language).

Como paso intermedio entre la arquitectura inicial de la figura 1 y la final de la figura 2, este proyecto propone la arquitectura mostrada en la figura 3. La diferencia entre esta arquitectura intermedia y la final es el modo de acceso al repositorio de objetos.

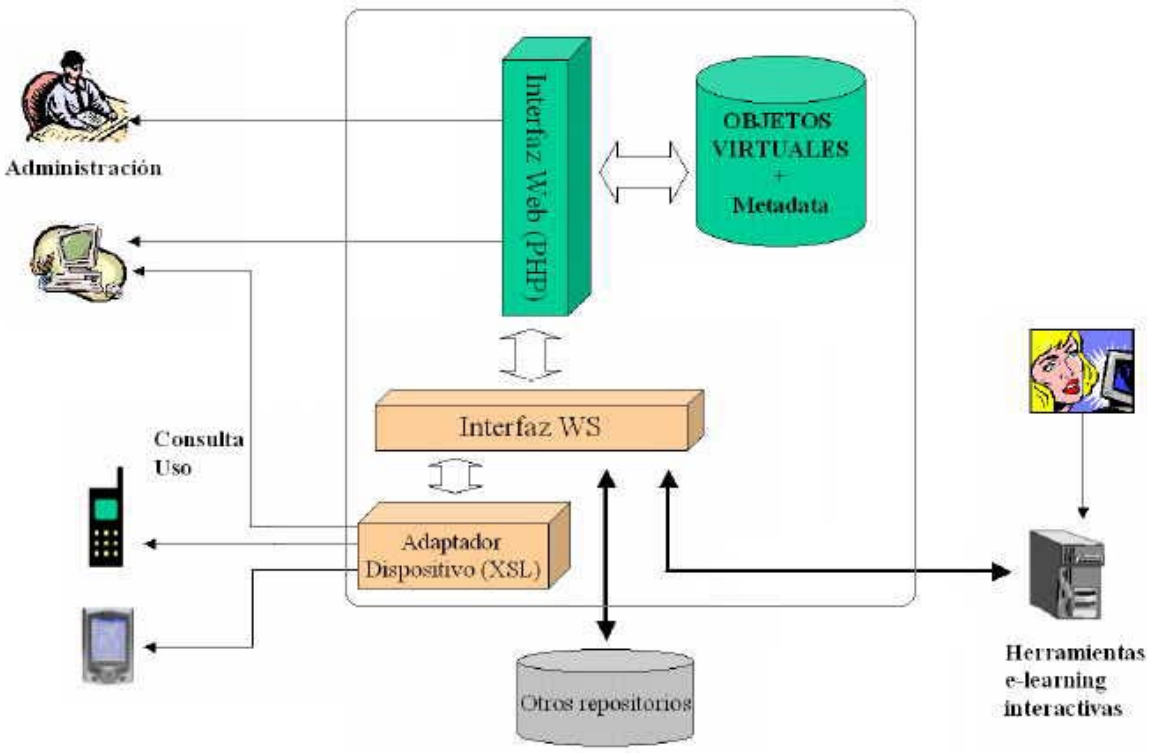

Figura 3. Arquitectura intermedia propuesta en el proyecto.

En la arquitectura final el acceso se propone de forma directa al repositorio almacenado en una base de datos, devolviendo como respuesta el procesamiento de determinadas consultas a esa base de datos.

En la arquitectura intermedia el acceso al repositorio se realiza interactuando con la interfaz web de usuario para obtener los datos requeridos. La comunicación con el repositorio se realiza mediante solicitudes HTTP de tipo POST y GET [12]. El Servicio Web ejecuta una operación mediante peticiones HTTP *request* en base a los parámetros de la operación y recupera los datos de la respuesta HTTP *response*. Por tanto, esta primera versión de la interfaz del Servicio Web se ha desarrollado como un adaptador desplegado por encima de la interfaz web de los museos virtuales MIGS y CHASQUI.

En dichas solicitudes HTTP se simulan operaciones que ofrece la interfaz web. Como respuesta se obtiene una página HTML (al igual que al realizar la operación por

medio de un navegador) que se interpreta para obtener la información deseada y devolver la respuesta en formato XML.

El cambio de la arquitectura intermedia a la final no plantea problemas si se conserva el mismo interfaz ya que las aplicaciones seguirán accediendo del mismo modo y obtendrán la misma información, sólo cambiará la forma en que se obtiene la información, lo cuál se hace de forma transparente al visitante.

2. INTRODUCCIÓN A LOS SERVICIOS WEB

2.1. ¿QUÉ ES UN SERVICIO WEB?

Un Servicio Web es una interfaz que ofrece un conjunto de operaciones, a las cuales se accede mediante un protocolo de transporte adecuado, que suele ser HTTP, y que tiene como objetivo facilitar la interoperabilidad entre aplicaciones situadas en diferentes contextos.

Los Servicios Web permiten que las aplicaciones compartan información y que además invoquen funciones de otras aplicaciones independientemente de cómo se hayan creado, cuál sea el sistema operativo o la plataforma en que se ejecutan y cuáles sean los dispositivos utilizados para obtener acceso a ellas.

2.2. ¿POR QUÉ SERVICIOS WEB?

La principal cualidad de los Servicios Web es que permiten una conexión estandarizada entre sistemas y aplicaciones dispares a través de la red, favoreciendo la interoperabilidad.

En un futuro se podrá simplificar la integración de aplicaciones gracias a los Servicio Web, reducir los costes o automatizar los procesos de negocio.

Con los Servicios Web podemos reutilizar desarrollos ya utilizados sin importar la plataforma en la que funcionan o el lenguaje en el que están escritos. Los Servicios Web se constituyen en una capa adicional a estas aplicaciones de tal forma que pueden interaccionar entre ellas usando para comunicarse tecnologías estándares que han sido desarrolladas en el contexto de Internet.

Los Servicios Web son el siguiente paso lógico en la creación de una plataforma de tecnología abierta que permita la comunicación transparente entre las distintas aplicaciones, procesos, sistemas y datos. Con los Servicios Web, las empresas pueden interconectar a sus clientes, empleados, proveedores y socios, sin importar cuáles son las tecnologías propietarias que cada uno de ellos utilice. Es más, el software de Servicios Web permite a las empresas que sus sistemas de legado se incorporen dentro de esta infraestructura integrada sin necesidad de remplazarlos. Se trata de evolución más que revolución.

Las aplicaciones web actuales ya no son suficientes. El modelo actual de negocio electrónico no facilita la integración de las aplicaciones de Internet con el resto de software de las empresas. Si las compañías quieren extraer el máximo beneficio de Internet, los sitios web deben evolucionar. Este es el contexto en el que surgen los Servicios Web.

Un Servicio Web puede agregar otros Servicios Web para proveer un conjunto de características mejoradas. Por ejemplo, un Servicio Web puede proveer

características de alto nivel en viajes, y sirviéndose para ello de Servicios Web de bajo nivel de renta de automóviles, billetes de avión, y hoteles.

Las aplicaciones futuras se realizarán con Servicios Web que serán seleccionados dinámicamente en tiempo real basados en costo, calidad, y disponibilidad.

Otro punto relevante es que el cliente de una página web normalmente es un navegador controlado por un usuario humano, mientras que un Servicio Web está orientado a ser usado por programas (aplicaciones, navegadores, móviles, PDA, etc).

2.3. ÉXITOS DE LOS SERVICIOS WEB

Entre las razones por las cuales los Servicios Web jugarán un rol principal en la siguiente generación de sistemas distribuidos, están:

- *Interoperabilidad*. Cualquier Servicio Web puede interactuar con cualquier otro Servicio Web. El protocolo estándar SOAP (ver sección 2.5.) permite que cualquier servicio pueda ser ofrecido o utilizado independientemente del lenguaje o ambiente en que se haya desarrollado.

- *Omnipresencia*. Los Servicios Web se comunican utilizando HTTP y XML. Cualquier dispositivo que trabaje con éstas tecnologías puede por tanto ser huésped de, y acceder a, los Servicios Web. Por ejemplo, pronto serán utilizados en teléfonos, automóviles e incluso en máquinas vendedoras de refrescos. Por ejemplo, una máquina de venta de refrescos puede comunicarse vía inalámbrica con el Servicio Web de un proveedor local y ordenar un pedido de suministro.

- *Baja rampa de entrada para los desarrolladores*. Los conceptos subyacentes a los Servicios Web son fáciles de comprender. Así mismo existen multitud de herramientas de desarrollo (ToolKits) ofrecidos, entre otros, por IBM, Sun Microsystems y la organización de Apache, que permiten a los desarrolladores crear e implementar rápidamente Servicios Web.

- *Apoyo de las Industrias*. Todas las compañías apoyan el protocolo SOAP y la tecnología derivada de los Servicios Web.

2.4. EJEMPLOS DE USO DE SERVICIOS WEB

Por ejemplo una empresa fabricante de un producto X podría usar un Servicio Web para permitir la integración de su módulo de inventario con el módulo de disponibilidad de los sistemas de distribuidores y subdistribuidores, todo de una manera simple y sin importar que plataformas utilicen.

Otro ejemplo, sería el caso de una sociedad que agrupe a las empresas de comercio exterior, que este encargada de mantener actualizados parámetros tales como el tipo de cambio, y que a través de un Servicio Web permita la integración de está información en los sistemas de las empresas.

Un ejemplo real, es el servicio de traducción, que ofrece un conocido buscador de páginas web, este servicio puede ser accedido también a través de un Servico Web, de manera que únicamente tenemos que invocar el servicio para tener integrado un traductor de varios idiomas en nuestras aplicaciones.

2.5. MODELO GENERAL DE LOS SERVICIOS WEB

La misma información, en formato XML, estará disponible para aplicaciones escritas en un lenguaje de programación, otros Servicios Web, o incluso navegadores de usuario final, como se muestra en la figura 4. En cada caso, será necesario implementar un cliente del Servicio Web, que será el responsable de controlar la interacción entre el servidor y el cliente, generando los mensajes XML necesarios.

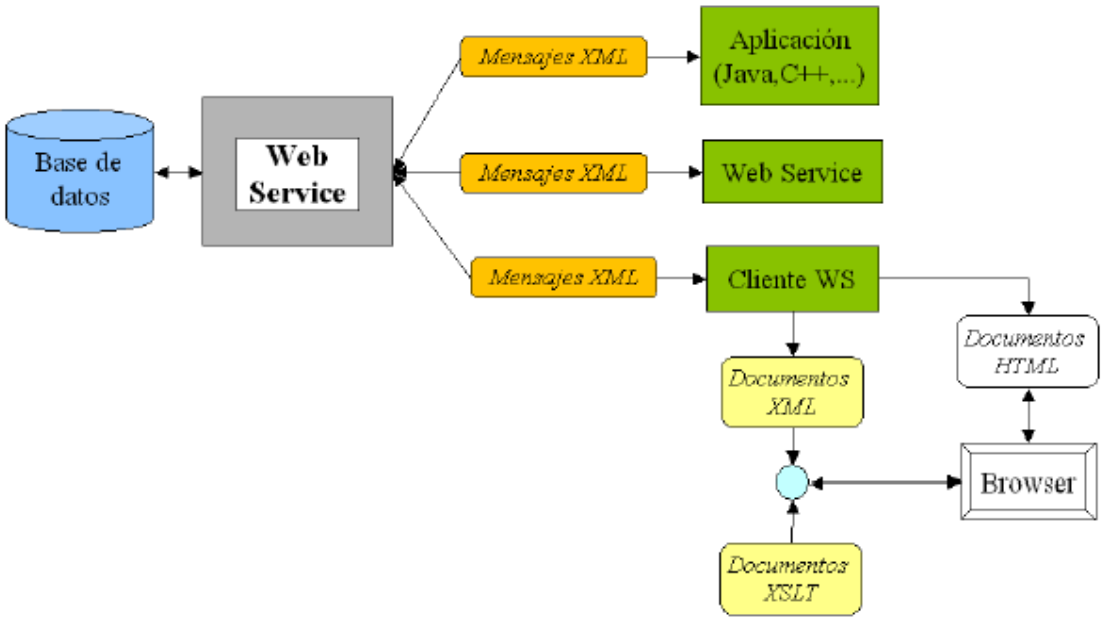

Figura 4. Modelo general de Servicio Web.

2.6. ARQUITECTURA DE LOS SERVICIOS WEB

Como podemos apreciar en la figura 5, la arquitectura de Servicios Web es sencillamente un envoltorio para acceder a un código preexistente de manera independiente del lenguaje y de la plataforma.

Es una meta-arquitectura que permite que ciertos servicios de red sean dinámicamente descritos, publicados, descubiertos e invocados en un ambiente de cómputo distribuido.

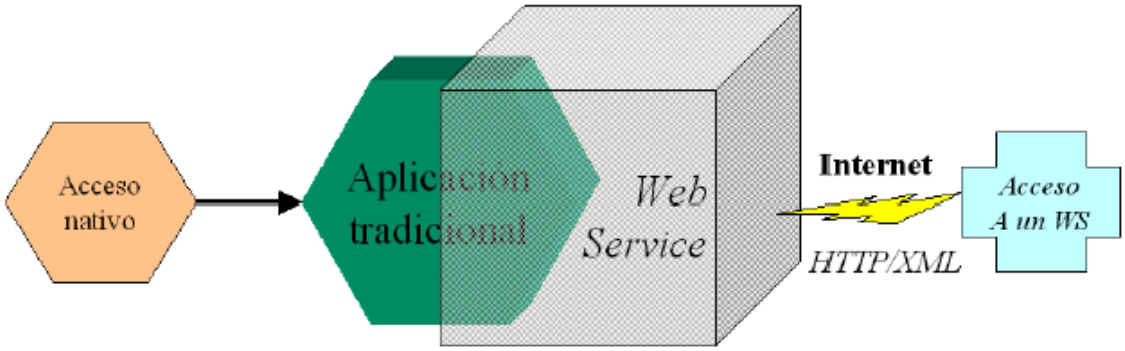

Figura 5. Servicio Web como *nueva interfaz* de una aplicación.

2.6.1. PILA CONCEPTUAL DE LOS SERVICIOS WEB

Los Servicios Web definen un nuevo modelo de construcción de aplicaciones distribuidas, basadas en componentes. Esta arquitectura debe estar claramente definida y asentada en estándares en todas sus capas, que permitan garantizar la interoperabilidad entre sistemas heterogéneos, el objetivo principal de esta tecnología.

La base común a las diferentes visiones de la arquitectura de Servicios Web conforma la pila de protocolos que se muestra en la figura 6.

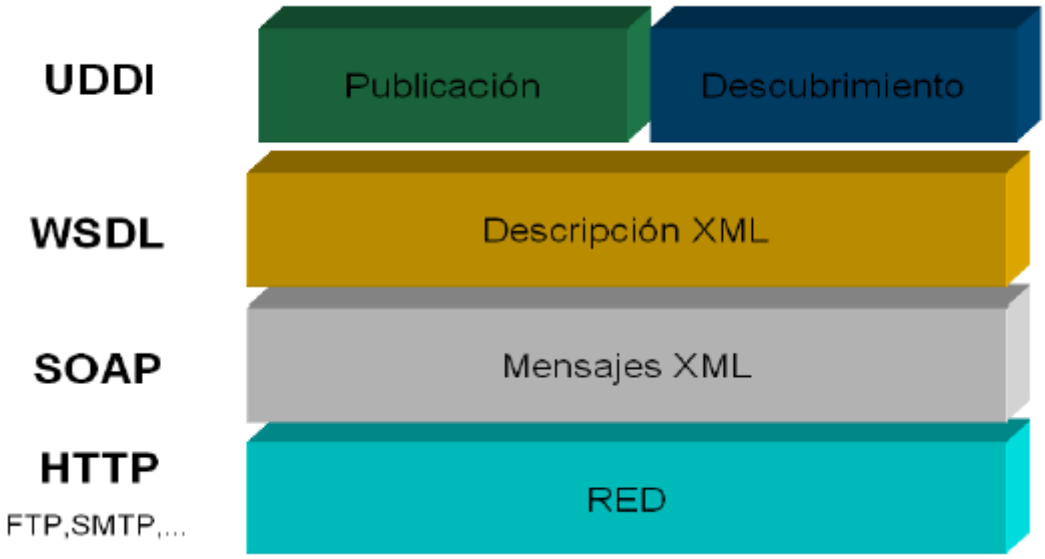

Figura 6. Pila Conceptual de los Servicios Web

La base es la *capa de red*, ya que la característica esencial de un Servicio Web es su carácter distribuido y remoto. HTTP es el protocolo *de facto*, gracias a su amplia difusión, si bien podría asentarse en otros como MQSeries (message-queuing system), FTP (File Transfer Protocol) [13], SMTP (Simple Mail Transfer Protocol), RMI/IIOP (Remote Method Invocation over Internet Inter-Orb Protocol), etc. En todo caso, la capa de red se plantea como algo completamente transparente para el desarrollo de Servicios Web, dado su grado de independencia e interoperabilidad.

Por encima de la capa de red tendremos una *capa de comunicación* basada en un protocolo estándar e independiente de lenguajes concretos, por lo que se ha definido a partir de *XML* (eXtensible Markup Language) [14]. Para cubrir estos requisitos, surgió el estándar *SOAP* (Simple Object Access Protocol) , un protocolo ligero de intercambio de mensajes XML, que define un mecanismo estándar para llamadas y respuestas remotas a través de XML, con soporte para múltiples protocolos de red, y permite mantenerse aislado de los detalles de implementación de los Servicios Web en un cierto lenguaje de programación.

La capa de *descripción de servicios* pretende cubrir la fase de definición formal, para que otros accedan al servicio. Para cubrir esta funcionalidad existe *WSDL* (Web Services Description Language) [15], el estándar de definición formal de un Servicio Web. Basado en los XML Schemas, proporciona mecanismos para definir formalmente los servicios y así automatizar la generación de clientes de acceso a un Servicio Web. Un documento WSDL puede complementarse con otros documentos de descripción de servicios para especificar los aspectos de alto nivel, como son el contexto de negocio, la calidad de servicio y las relaciones con otros Servicios Web.

En la capa superior se encuentra la *publicación* y *descubrimiento de los Servicios Web*. Un Servicio Web debe ser publicado para que terceros lo descubran. La forma más sencilla de hacerlo consiste en la publicación estática del fichero WSDL en el mismo servidor que oferta el servicio. Otra alternativa consiste en localizar el servicio

usando un registro *UDDI* (Universal Description Discovery and Integration) [16], el modelo de repositorio para Servicios Web, en el que se registran tanto las empresas e instituciones, así como los Servicios Web que ofrecen, organizados en taxonomías.

2.6.2. ROLES EN EL MODELO DE SERVICIOS WEB

La arquitectura del modelo tradicional definido para los Servicios Web plantea, en el caso más general, la existencia de tres roles que se muestran en la figura 7.

Figura 7. Roles en el modelo de Servicios Web.

El *proveedor del servicio* es el punto que posee el servicio, es quien lo ha desarrollado y lo ha puesto en explotación para ser utilizado por otros. Es el responsable de proporcionar un entorno de ejecución válido, así como una especificación del acceso y utilización del mismo. Desde el punto de vista comercial, es quien presta el servicio. Desde el punto de vista de arquitectura, es la plataforma que provee el servicio.

El *solicitante del servicio* representa al usuario del servicio, es quien lo utiliza, según las funcionalidades disponibles del mismo. Este solicitante puede ser: un usuario final, un programa que accede al servicio, u otro Servicio Web, ya que una de las características fundamentales es la posibilidad de componer Servicios Web para desarrollar sistemas más complejos. Desde el punto de vista comercial, la empresa que requiere cierto servicio. Desde el punto de vista de la arquitectura, la aplicación o cliente que busca e invoca un servicio.

El *registrador del servicio* es un rol opcional, ya que la publicación de la especificación de un servicio puede hacerse localmente al proveedor del servicio, de forma que se encuentre disponible en la misma ubicación que el servicio en sí. La idea de este rol es la de proporcionar un punto común donde publicar los servicios, y que puede ser utilizado por otros como un catálogo de servicios. Es un depósito de descripciones de servicios que puede ser consultado, donde los proveedores de servicios publican sus servicios y los solicitantes encuentran los servicios y detalles para utilizar dichos servicios.

2.7. IMPLEMENTACIÓN DE UN SERVICIO WEB

En la figura 8 se presenta un breve esquema paso a paso de cómo se implementa un Servicio Web.

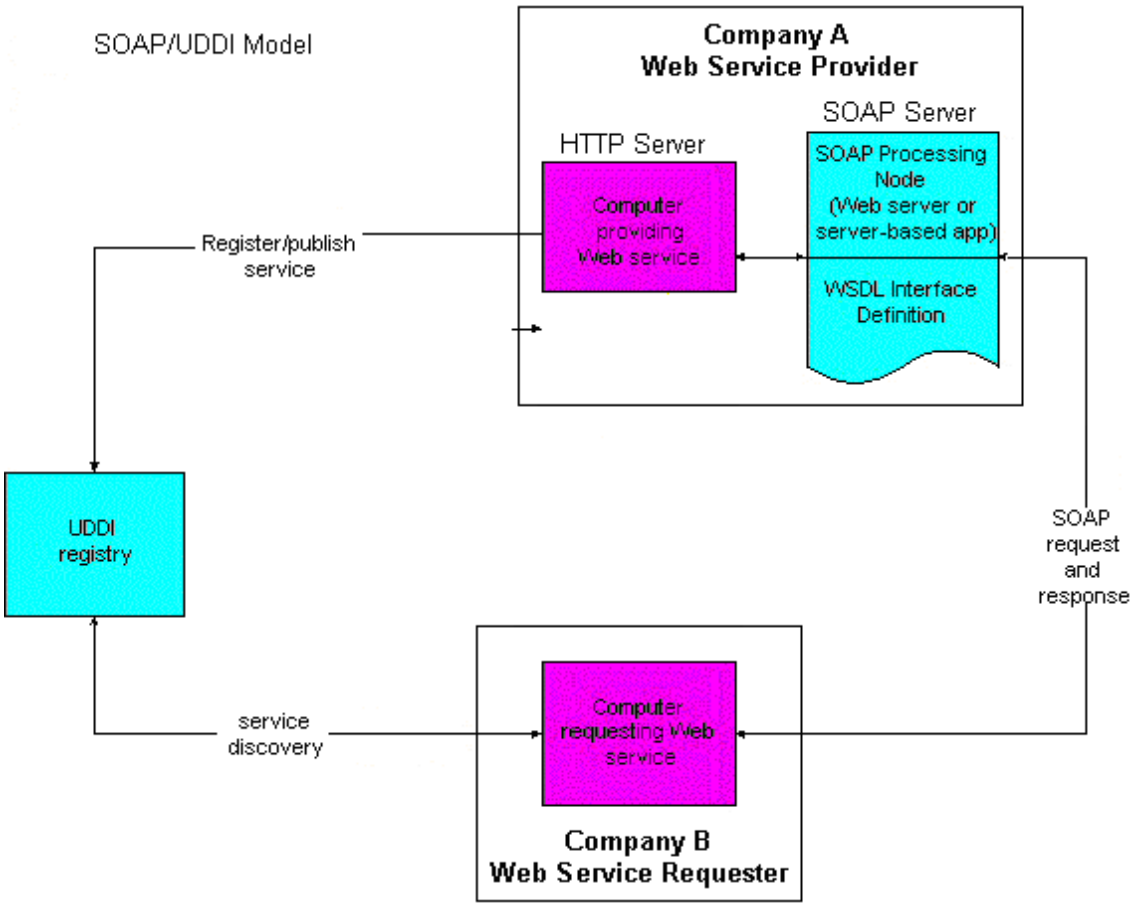

Figura 8. Modelo SOAP/UDDI de implementación de Servicios Web

1. Un proveedor de servicio crea un Servicio Web.

2. El proveedor del servicio utiliza WSDL para describir el servicio (a un registro UDDI).

3. El proveedor del servicio registra el servicio en un registro UDDI.

4. Otro servicio o usuario localiza y solicita el servicio registrado al consultar los registros UDDI

5. El servicio o usuario solicitante escribe una aplicación que liga el servicio registrado utilizando SOAP.

6. Se intercambian datos y mensajes XML sobre HTTP.

2.8. DESCRIPCIÓN DE LENGUAJES Y PROTOCOLOS

Los Servicios Web se registran y anuncian utilizando los servicios y protocolos que se muestran en la tabla 1. Muchos de estos estándares y otros están siendo desarrollados en el proyecto UDDI, un consorcio de industrias que coordina los esfuerzos de diseño y creación.

XML eXtensible Markup Language	Un Servicio Web es una aplicación web creada en XML.
WSDL Web Services Definition Service	Este protocolo se encarga de describir el Servicio Web cuando es publicado. Es el lenguaje XML que los proveedores emplean para describir sus Servicios Web.
SOAP Simple Object Access Protocol	Permite que programas que corren en diferentes sistemas operativos se comuniquen. La comunicación entre las diferentes entidades se realiza mediante mensajes que son rutados en un sobre SOAP.
UDDI Universal Description Discovery and Integration	Este protocolo permite la publicación y localización de los servicios. Los directorios UDDI actúan como una guía telefónica de Servicios Web.

Tabla 1. Lenguajes y protocolos empleados por el Servicio Web

2.8.1. XML (EXTENSIBLE MARKUP LANGUAGE)

Se inició en Febrero de 1998 y ha revolucionado la forma en que estructuramos, describimos e intercambiamos información. Independientemente de múltiples formas en que se utiliza hoy en día XML, todas las tecnologías de Servicios Web se basan en XML. El diseño de XML se deriva de dos fuentes principales: SGML (Standard Generalized Markup Language) [17] y de HTML (HyperText Markup Language) [18].

2.8.2. SOAP (SIMPLE OBJECT ACCESS PROTOCOL)

Es un protocolo para iniciar las conversaciones con un servicio UDDI. SOAP simplifica el acceso a los objetos, permitiendo a las aplicaciones invocar métodos objeto o funciones, que residen en sistemas remotos. Una aplicación SOAP crea una petición bloque en XML. proporcionando los datos necesarios para el método remoto así como la ubicación misma del objeto remoto.

SOAP es un lenguaje de mensajería basada en XML, estandarizado por el consorcio W3C, especifica todas las reglas necesarias para ubicar Servicios Web XML, integrarlos en aplicaciones y establecer la comunicación entre ellos.

Un mensaje SOAP se compone de un sobre que contiene el cuerpo del mensaje y cualquier información de cabecera que se utilice para describir el mensaje. A continuación vemos un ejemplo en la figura 9:

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
    <!--Optional header information goes here. -->
    <To>Scott</To>
    <From>Suzanne</From>
  </soap:Header>
  <soap:Body>
    <!--Message goes here. -->
    Please pick up some milk on your way home from work.
  </soap:Body>
</soap:Envelope>
```

Figura 9: Anatomía de un mensaje SOAP

En la figura 10 vemos el funcionamiento de SOAP:

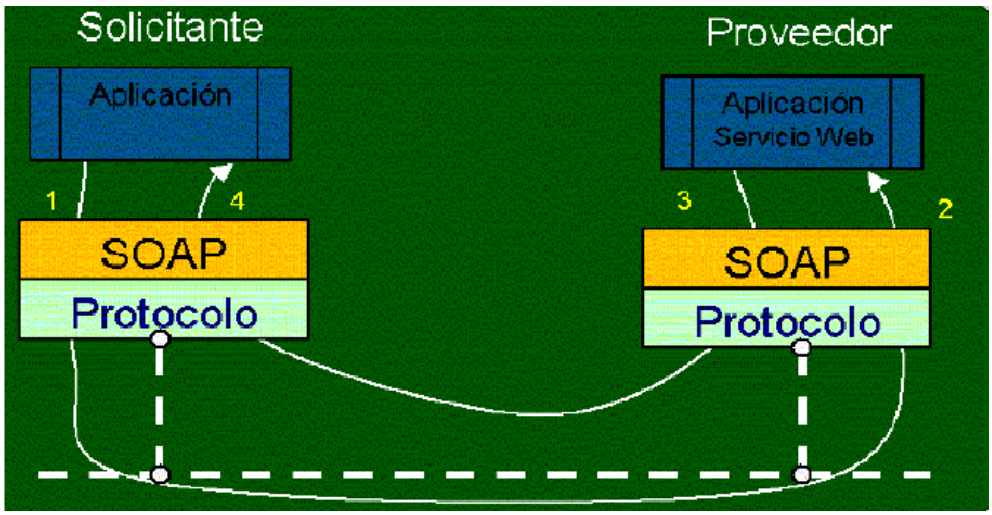

Figura 10. Comunicación en SOAP

1. El cliente crea un mensaje SOAP, la petición que invoca al Servicio Web. El intérprete SOAP del cliente interactúa con el protocolo de red para enviar el mensaje a través de la red, encapsulada en el formato XML de SOAP.

2. El mensaje llega hasta el intérprete SOAP del servidor, que encamina la petición al Servicio Web. El intérprete SOAP se encarga de realizar las conversiones necesarias en objetos específicos del lenguaje de programación (marshall / unmarshall).

3. El Servicio Web procesa la petición y genera un resultado, que se convierte en la respuesta SOAP, que se envía de vuelta por la red.

4. El mensaje es recibido, y convertido a los tipos específicos de la aplicación del cliente.

XML-RPC (eXtensible Markup Language-Remote Procedure Call) [19] es una alternativa más sencilla a SOAP para la creación de Servicios Web basada en la idea de llamada a procedimientos remotos.

2.8.3. SOAP WITH ATTACHMENTS

Incluso aunque SOAP es un protocolo basado en XML, no siempre es conveniente forzar que todos los datos que se pasan como parte de un mensaje SOAP estén codificados de una forma que pueda ser embebida dentro de un documento XML. Algunas veces sería mejor sólo permitir que datos, como imágenes JPEG, ficheros WAV, etc., sean pasados junto con el mensaje SOAP en vez de dentro de él. Para corregir este problema, se creó la especificación *SOAP Messages with Attachments* [20]. Esta especificación permite embeber un mensaje SOAP dentro de un documento MIME (Multi-Purpose Internet Mail Extensions) [21], de una manera en la que no altere la reglas de procesamiento del mensaje. La especificación también describe cómo referirse a los attachments desde dentro del cuerpo de un mensaje SOAP.

2.8.4. WSDL (WEB SERVICE DESCRIPTION LANGUAGE)

Es el estándar propuesto para la descripción de los Servicios Web, el cual consiste en un lenguaje de definición de interfaz (IDL - Interface Definition Language) dels servicio basado en XML, que define la interfaz del servicio y sus características de implementación. El WSDL es apuntado en los registros UDDI y describe los mensajes SOAP que definen un Servicio Web en particular.

WSDL es una especificación que permite especificar formalmente un Servicio Web. Un documento WSDL describe , de forma independiente, la funcionalidad abstracta ofrecida por un servicio, y los detalles concretos de implementación (cómo y dónde).

WSDL se sitúa en la capa de descripción de Servicios Web. Es un modelo de definición general de Servicios Web, y por tanto no está necesariamente asociado a SOAP. Aún así, ya existen enlaces para describir Servicios Web que estén basados en SOAP que es el estándar de facto para la comunicación entre servicios.

WSDL está basado en los XML Schemas [22]. Los XML Schemas son un formalismo para definir la estructura de los documentos XML. Son la evolución de las DTD (Document Type Definition) [23], y actualmente constituyen el estándar para la descripción formal de documentos XML, dada su potencia descriptiva, que incorpora capacidades para definir gramáticas con soporte para herencia, restricciones sobre tipos, extensibilidad, etc.

Los Servicios Web se fundamentan en una capa de mensajería XML entre cliente y servidor. El mecanismo obvio para formalizar dicha comunicación sería, por tanto, los XML Schemas. Pero existen ciertos criterios de semántica (qué elementos son los métodos invocados, cuáles los argumentos, dónde enlazar para hacer la petición,

etc.) que no podrían ser cubiertos directamente por los XML Schemas. Y son esos aspectos los que complementa el estándar WSDL.

WSDL describe los Servicios Web como colecciones de puntos finales de comunicación capaces de intercambiar mensajes. Un documento WSDL es como un contrato entre el servidor y el cliente: el servidor se compromete a proveer ciertos servicios sólo si el cliente manda una petición con el formato adecuado.

En un fichero WSDL se describen los servicios con el siguiente modelo:

- Los *tipos* permiten definir tipos complejos, mediante XML Schemas, que podrán ser usados en el resto del documento WSDL.

- Los *mensajes* son los datos que se intercambian entre cliente y servidor. Cada uno de estos mensajes están formados por una o más partes, que definen la estructura de los datos intercambiados; cada una de estas partes serán de un determinado tipo (simple o definido para ese servicio).

- Las *operaciones* ofrecidas por un Servicio Web son flujos de *mensajes* de entrada/salida que se intercambian entre dos extremos (cliente y servidor). Las operaciones se agrupan en un tipo de puerto (colección de operaciones ofrecidas por un Servicio Web).

- Un *tipo de puerto* se enlaza con un protocolo de mensajería XML concreto (por ejemplo SOAP).

- Por último, se definen los puntos finales de acceso al Servicio Web *(puerto*), mediante un localizador (por ejemplo una URL). La agrupación de varios puntos de acceso definen un *servicio*.

A lo largo del documento WSDL existen referencias a partes previas del mismo (las partes de los mensajes hacen uso de tipos previos, las operaciones se forman mediante mensajes, etc.) En todos los casos, estas referencias se hacen mediante atributos de los elementos: cada descripción en el documento WSDL lleva un atributo *name* que la identifica (con el espacio de nombres correspondiente). Cuando desde otra parte del documento WSDL se referencia a esa descripción, se empleará otro atributo (message, type, etc.), cuyo valor se corresponderá con el nombre del elemento referenciado.

Si se examina cada parte de un documento WSDL se encontrará:

- <*definitions*>. El elemento <definitions> contiene la definición de uno o más servicios. En la mayoría de los casos, un archivo WSDL define un servicio únicamente. Seguido de la etiqueta de definición se encontrarán declaraciones de algunos atributos. Dentro de la etiqueta <definitions> se encuentran tres secciones conceptuales:

    - <*message*>. Definen el formato de los bloques de datos que van a ser intercambiados, de entrada y salida del servicio. Cada mensaje estará formado por una o varias partes (argumentos de los métodos invocados,

o resultado de dicho método), donde cada parte se corresponderá con un tipo básico o uno definido en el mismo documento WSDL. Los mensajes constituyen bloques abstractos de información, sin un uso específico como entrada o salida de un método.

- <*portType*>. Los tipos de puerto agrupan un conjunto de operaciones, son equivalentes a clases de programación que engloban diferentes métodos. Cada operación representa la funcionalidad ofrecida por el Servicio Web, y estará especificada mediante un intercambio de información entrante y saliente, donde cada uno de estos bloques se corresponderán con mensajes definidos anteriormente; en este punto se especifica la semántica de los mensajes: argumentos (input) o respuestas (output) de los métodos.

- <*binding*>. Los enlaces permiten asociar un tipo de puerto con un protocolo concreto de transporte y una codificación para los datos que se intercambian. Ya que WSDL no se limita a describir servicios SOAP, podría emplearse para indicar enlaces sobre otros protocolos de mensajería. En el caso de utilizar SOAP, se indicará el uso de SOAP como protocolo de mensajería XML. El protocolo de transporte (típicamente HTTP) y la codificación de los datos se especifica dentro del cuerpo de los mensajes SOAP (mensajes definidos como input/output para cada una de las operaciones).

- **<***service*>. Es la agrupación de uno o varios puertos, donde conectarse al servicio. Es importante destacar que un mismo servicio puede encontrarse disponible en diferentes ubicaciones (puertos) o accesible mediante diferentes protocolos (en este último caso, existirían definiciones diferenciadas a nivel de enlace).

- <*documentation*>. Cualquier elemento WSDL puede contener información del servicio para el usuario.

Como ejemplo se muestra el siguiente documento wsdl en la figura 11:

```
<?xml version="1.0" encoding="UTF-8" ?>
<definitions name="FooSample"
targetNamespace="http://tempuri.org/wsdl/"
xmlns:wsdlns="http://tempuri.org/wsdl/"
xmlns:typens="http://tempuri.org/xsd"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:stk="http://schemas.microsoft.com/soap-toolkit/wsdl-extension"
xmlns="http://schemas.xmlsoap.org/wsdl/">

<types>
<schema targetNamespace="http://tempuri.org/xsd"
xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
elementFormDefault="qualified" >
</schema>
</types>

<message name="Simple.foo">
  <part name="arg" type="xsd:int"/>
</message>
<message name="Simple.fooResponse">
  <part name="result" type="xsd:int"/>
</message>

<portType name="SimplePortType">
  <operation name="foo" parameterOrder="arg" >
  <input message="wsdlns:Simple.foo"/>
  <output message="wsdlns:Simple.fooResponse"/>
  </operation>
</portType>

<binding name="SimpleBinding" type="wsdlns:SimplePortType">
 <stk:binding preferredEncoding="UTF-8" />
 <soap:binding style="rpc"
       transport="http://schemas.xmlsoap.org/soap/http"/>
 <operation name="foo">
  <soap:operation
  soapAction="http://tempuri.org/action/Simple.foo"/>
  <input>
   <soap:body use="encoded" namespace="http://tempuri.org/message/"
   encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
  </input>
  <output>
   <soap:body use="encoded" namespace="http://tempuri.org/message/"
   encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
  </output>
 </operation>
</binding>
<service name="FOOSAMPLEService">
 <port name="SimplePort" binding="wsdlns:SimpleBinding">
 <soap:address location="http://carlos:8080/FooSample/FooSample.asp"/>
 </port>
</service>
</definitions>
```

Definición de tipos

Definición de 2 mensajes

Definición de 1 Port Type

Enlace sobre SOAP

El servicio, con un único punto de acceso (port), indicado como una url

Figura 11. documento WSDL

2.8.5. UDDI (UNIVERSAL DESCRIPTION, INTEGRATION)

Es una norma para describir los componentes disponibles de Servicios Web. Este estándar permite a las empresas registrarse en un tipo de directorio, sección

amarilla , de Internet que les ayuda a anunciar sus servicios y ofrece una clasificación de estos tanto en función de los servicios como de las empresas que los ofrecen, de tal forma que las compañías puedan encontrarse unas a otras y realizar transacciones en la web. El proceso de registro y consultas se realiza utilizando mecanismos basados en XML y HTTP(S). En el proyecto UDDI se trabaja para proveer un método de acceso común a los metadatos necesarios para determinar si un elemento de código previamente elaborado es suficiente, y si lo es, cómo accederlo.

Una vez implementado un Servicio Web y definido un documento WSDL para especificar formalmente cómo utilizar el Servicio, es posible confiar en que todos los posibles clientes encuentren directamente esta especificación, o bien se puede hacer uso de un repositorio UDDI para publicar dicho servicio.

UDDI es un modelo de registro público que permite a los desarrolladores de servicios publicar sus Servicios Web, y a los consumidores encontrar aquellos Servicios que más le interesen. Evidentemente, una organización o particular puede desempeñar ambos roles simultáneamente.

El modelo de funcionamiento de UDDI puede verse en la figura 12. Se plantea la existencia de un repositorio de servicios y de empresas, que es poblado con nuevas definiciones. Estas definiciones son consultadas por los motores de búsqueda o empresas finales, para localizar el servicio deseado, y poder comenzar a utilizarlo. Este proceso es cíclico.

Figura 12. Modelo UDDI

Las especificaciones UDDI consisten en un modelo de repositorio y un API para el registro y consulta de los Servicios Web, disponible sobre SOAP. Dentro de este repositorio se contempla la existencia de información sobre:

Organizaciones:

- Páginas blancas: direcciones, contactos e identificadores conocidos.

- Páginas amarillas: clasificaciones basadas en taxonomías estándar (localización y tipo de productos/servicios).

- Páginas verdes: información técnica sobre los servicios que se exponen.

Servicios Web:

- Enlace a la definición técnica del servicio (tModel), referenciando su documento de especificación formal (WSDL), mediante una URL externa.

- Colección de tModel, como puntos de acceso al Servicio (Binding Template), las ubicaciones donde conectarse para usar el servicio.

- El modelo de registros UDDI contempla la existencia de:

    - Registros *públicos*, empresas que ofrecen un punto de acceso público, donde se registran los servicios ofertados a terceros.

    - Registros *privados*, internos a las organizaciones, donde se registran los servicios de ámbito local.

No existe una relación formal entre UDDI y WSDL, ya que la especificación de los registros UDDI se realizó de forma que sea independiente del formalismo utilizado para la descripción final del Servicio. Aún así, la información para describir un servicio, proporcionada por un documento WSDL, complementa a la información que se puede encontrar en un registro UDDI.

Un error habitual es considerar que las descripciones completas de los Servicios se encuentran dentro del repositorio UDDI. No es así: en un registro UDDI **NO** se almacena la descripción WSDL de un Servicio, pero sí un enlace a donde se encuentra dicha descripción, así como una clasificación de dicho Servicio.

2.9. NUESTRO SERVICIO WEB

Este Servicio Web se crea para favorecer la conexión a dos museos virtuales existentes en la UCM: MIGS (Museo de Informática García Santesmases) y CHASQUI (Museo de Arqueología del Departamento de Historia de América II).

Se implementa como un recubrimiento a las aplicaciones web existentes para ambos museos de manera que se ofrece una nueva interfaz independiente de los lenguajes de programación en que dichas aplicaciones están desarrolladas.

En dicha implementación se emplea el modelo SOAP/UDDI mostrado en la figura 13.

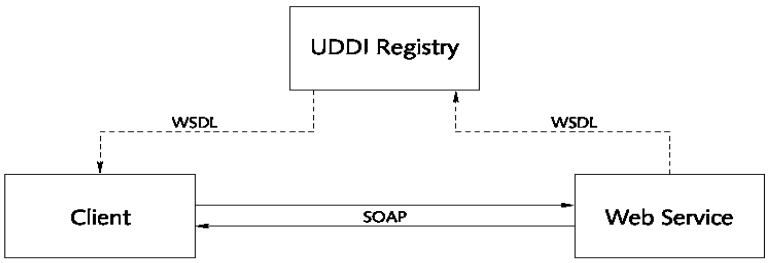

Figura 13. Modelo SOAP/UDDI

La descripción del servicio se realiza mediante WSDL y se emplaza en el registro UDDI, la aplicación de gestión de museos localiza el servicio mediante consultas al registro UDDI y se comunica con éste mediante mensajes SOAP.

Se apoya en las facilidades ofrecidas por el proyecto Apache, AXIS [24], que genera automáticamente el fichero de descripción del servicio WSDL, y a partir de este es capaz de generar código que proporciona la base de la implementación del Servicio Web y del cliente que accederá a este servicio.

Posteriormente, se verá el diseño del Servicio Web y se aclarará qué partes de dicho diseño corresponden a código generado automáticamente por Axis.

3. TECNOLOGÍAS Y HERRAMIENTAS UTILIZADAS

En este apartado se comentan las diferentes herramientas y tecnologías empleadas en el desarrollo de este proyecto:

3.1. LENGUAJE DE PROGRAMACIÓN: JAVA

A continuación se comentan los motivos por los que se decidió emplear Java [25] para la implementación de este proyecto.

- Java es un *lenguaje orientado a objetos*. Esto significa que posee ciertas características como:
    - Objetos
    - Clases
    - Métodos
    - Subclases
    - Herencia simple
    - Enlace dinámico
    - Encapsulamiento
- La ventaja potencial de Java se encuentra en sus *bibliotecas de clases*. Una biblioteca de clases es mucho más fácil de usar que una biblioteca de procedimientos, como las de C. Esto se debe a que las clases ofrecen mecanismos de abstracción más eficaces que los procedimientos.
- Java elimina las complejidades de otros lenguajes, tales como manejo de memoria dinámica controlada desde el programa.
- Es multiplataforma.
- Robustez, los errores se detectan en el momento de producirse, lo que facilita la depuración.

Se emplea Java 2 Platform Standard Edition 5.0 pues posee el API más actualizado.

3.2. TOMCAT

Tomcat (Jakarta Tomcat) [26] funciona como un contenedor de servlets desarrollado bajo el proyecto Jakarta en la Apache Software Foundation. Tomcat implementa las especificaciones de los servlets y JSPs de Sun Microsystems. Se le considera un servidor de aplicaciones.

Debido a que se emplean servlets y JSP, y al uso de Axis, que trabaja sobre Tomcat, se decidió que Tomcat fuese el servidor de aplicaciones de este proyecto.

En la tabla 2 se muestra la estructura de directorios de Tomcat.

Nombre de Directorio	Descripción
bin	Contiene los scripts de arrancar/parar
conf	Contiene varios ficheros de configuración incluyendo server.xml (el fichero de configuración principal de Tomcat) y web.xml que configura los valores por defecto para las distintas aplicaciones desplegadas en Tomcat.
doc	Contiene varia documentación sobre Tomcat
lib	Contiene varios ficheros jar que son utilizados por Tomcat.
logs	Aquí es donde Tomcat sitúa los ficheros de diario.
src	Los ficheros fuentes del API Servlet.
webapps	Contiene las aplicaciones web.

Tabla 2. Estructura de directorios de Tomcat.

Adicionalmente podemos tener los directorios de la tabla 3.

Nombre de Directorio	Descripción
Work	Generado automáticamente por Tomcat, este es el sitio donde Tomcat sitúa los ficheros intermedios (como las páginas JSP compiladas) durante su trabajo. Si borramos este directorio mientras se está ejecutando Tomcat no podremos ejecutar páginas JSP.
Classes	Podemos crear este directorio para añadir clases adicionales al classpath. Cualquier clase que añadamos a este directorio encontrará un lugar en el classpath de Tomcat.

Tabla 3. Directorios adicionales de Tomcat.

La versión empleada en este proyecto ha sido Tomcat 5.

3.3. SERVLETS

Los Servlets [27] son las respuesta de la tecnología Java a la programación CGI (Common Gateway Interface). Son programas que se ejecutan en un servidor web y construyen páginas web dinámicas. Construir páginas web al vuelo es útil por un número de razones.

En este proyecto las páginas web se basan en datos que envía el usuario y además su aspecto es cambiante dependiendo de los objetos virtuales que se encuentren en los museos de origen.

Ventajas de los Servlets:

Los Servlets Java son más eficientes, fáciles de usar, más poderosos, y más portables que el CGI tradicional y otras muchas tecnologías del tipo CGI.

- *Eficiencia*. Con los Servlets, la máquina virtual Java permanece arrancada. No es necesario arrancar un nuevo proceso con cada solicitud HTTP, y cada petición es manejada por un thread Java de peso ligero, no un pesado proceso del sistema operativo. Con los Servlets, si hay N procesos hay N threads pero sólo una copia de la clase Servlet.

- *Conveniencia*. Emplea un lenguaje familiar como Java. Los Servlets tienen una gran infraestructura para análisis automático y decodificación de datos de formularios HTML, leer y seleccionar cabeceras HTTP, manejar cookies, seguimiento de sesiones, y muchas otras utilidades.

- *Potencia*. Los servlets se comunican directamente con el servidor web. Pueden compartir los datos entre ellos. Y también pueden mantener información de solicitud en solicitud, simplificando aspectos como seguimiento de sesión y el caché de cálculos anteriores.

- *Portable*. Los servlets están escritos en Java y siguen un API bien estandarizado.

Por todas estas razones se eligieron los servlets para implementar el cliente de nuestro Servicio Web.

3.4. JSP

Java Server Pages (JSP) es una tecnología que permite mezclar HTML estático con HTML generado dinámicamente. Los servlets, hacen que se genere la página completa mediante el programa, incluso aunque la mayoría de ella sea siempre lo mismo. JSP nos permite crear dos partes de forma separada.

Ventajas de JSP:

- La parte dinámica está escrita en Java, por lo que es eficiente y fácil de usar.
- Portable a otros sistemas operativos y servidores web.
- La conveniencia de escribir y modificar HTML normal, en lugar de tener gran cantidad de sentencias println que generen HTML, como ocurre en el caso de los servlets.
- Actúa separando el formato del contenido.
- Permite ser usado de forma combinada con servlets.

La necesidad de usar JSP surgió cuando se vio que la presentación del formulario de búsqueda de objetos dependía del tipo de objetos virtuales que hubiese en cada momento en el repositorio del museo, lo que dio lugar a la necesidad de crear dinámicamente esas páginas web.

3.5. AXIS

Apache Axis es una implementación de SOAP (Simple Object Access Protocol)

Proporciona librerías necesarias para la generación de Servicios Web, un compilador de Java a WSDL y un compilador de WSDL a Java. Genera el documento WSDL correspondiente a un interfaz Java, dicha interfaz está sujeta a ciertas restricciones. El documento WSDL permite generar ficheros stubs (proxies) y skeletons para invocar e implementar Servicios Web. De esta manera un cliente escrito sobre cualquier plataforma pueda invocar el Servicio Web.

A continuación mediante la figura 14 mostramos como emplear Axis para generar el Servicio Web:

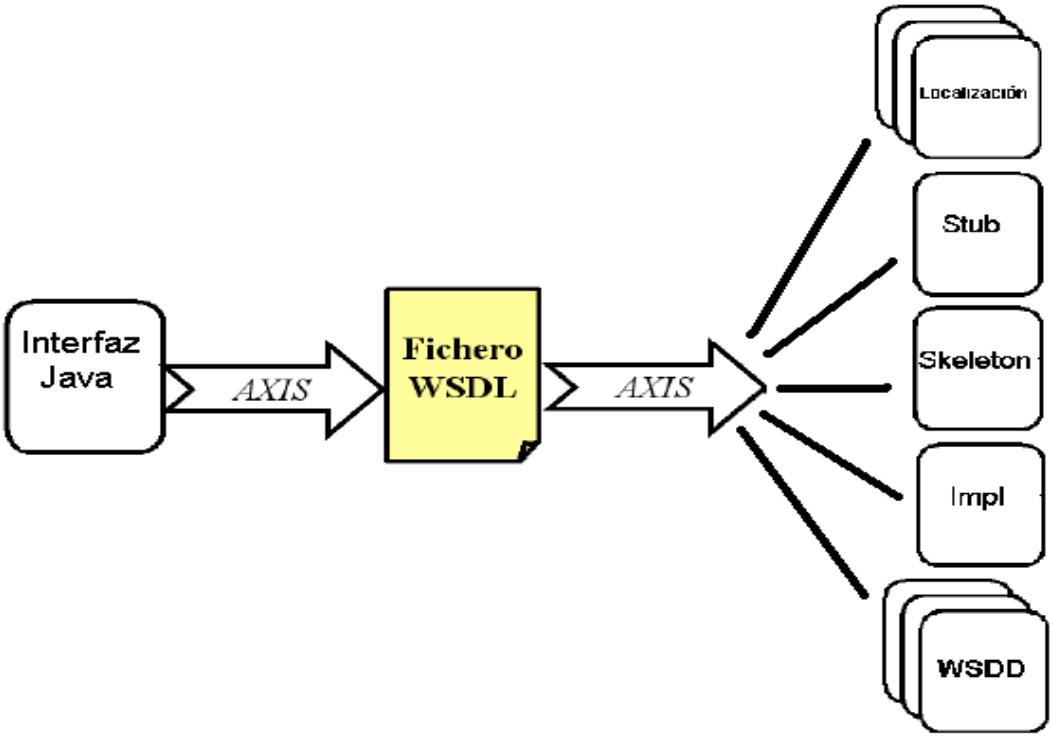

Figura 14. Funcionamiento de Axis

Pasos de generación de ficheros:

1. Crear la interfaz del Servicio Web.

2. A partir de esta interfaz, generar el archivo .wsdl.

3. Mediante este archivo, generar los ficheros del Servicio Web:

    - *Interfaz del Servicio Web* que contiene los métodos publicados por este. Esta interfaz se empleará tanto en el lado del cliente como en el lado del Servidor.

    - *Archivos de localización:* una interfaz y una clase que la implementa. Ofrecen métodos para la localización del Servicio Web a partir del punto de enlace indicado en el fichero de descripción .wsdl, u otros indicados cómo parámetros. Actúan del lado del cliente como factoría para objetos que representan al Servicio Web.

    - *Archivo Stub* para la codificación-decodificación entre los métodos y objetos Java y las peticiones-respuesta SOAP. Se emplea de forma transparente en clientes del Servicio Web siguiendo el patrón proxy.

    - *Archivo Impl* que ha de albergar la implementación de las operaciones ofrecidas por el servicio y por tanto el archivo generado automáticamente no contiene más que la cabecera de los métodos. Además de esto en este

fichero tiene lugar la codificación-decodificación de objetos Java y peticiones-respuesta SOAP del lado del servidor.

- *Archivo Skeleton* para la conexión del Servicio Web con el motor de Axis. Contiene la definición de cada método publicado. Actúa del lado del servidor realizando una llamada al método correspondiente del archivo Impl cada vez que se invoca una operación del Servicio Web.

- *Archivos .wsdd*, uno para publicar el Servicio Web y otro para eliminar el Servicio Web.

4. Publicar el Servicio Web mediante la ejecución del archivo .wsdd correspondiente.

3.5.1. CSS(CASCADING STYLE SHEETS)

El lenguaje HTML está limitado a la hora de aplicarle forma a un documento. Esto es así porque fue concebido para otros usos (científicos sobre todo), distintos a los actuales, mucho más amplios.

Las hojas de estilo intentan separar en un documento el estilo lógico del estilo físico, dejando este último en bloques de definición de estilos separados de la estructura del documento. La figura 15 muestra un modelo básico de cómo funcionan.

Figura 15. Modelo del funcionamiento de las hojas de estilo.

Las ventajas de utilizar CSS (u otro lenguaje de estilo) son:

- Control centralizado de la presentación de un sitio web completo con lo que se agiliza de forma considerable la actualización del mismo.

- Los navegadores permiten a los usuarios especificar su propia hoja de estilo local que será aplicada a un sitio web remoto, con lo que aumenta considerablemente la accesibilidad. Por ejemplo, personas con deficiencias visuales pueden configurar su propia hoja de estilo para aumentar el tamaño del texto o remarcar más los enlaces.

- Una página puede disponer de diferentes hojas de estilo según el dispositivo que la muestre o incluso a elección del usuario. Por ejemplo, para ser impresa, mostrada en un dispositivo móvil, o ser leída por un sintetizador de voz.

- El documento HTML en sí mismo es más claro de entender y se consigue reducir considerablemente su tamaño.

Hay varias versiones: CSS1 y CSS2, con CSS3 en desarrollo por el W3C (World Wide Web Consortium). Los navegadores modernos implementan CSS1 bastante bien, aunque existen pequeñas diferencias de implementación según marcas y versiones de los navegadores. CSS2, sin embargo, está solo parcialmente implantado en los más recientes.

3.5.1.1. APLICACIÓN DE ESTILO A ETIQUETAS

Se puede asignar el estilo a una etiqueta concreta de HTML. Para ello, en la declaración de estilos escribimos la etiqueta y entre llaves, los atributos de estilo que deseemos, como por ejemplo:

```
body {
   background-color: #f0f0f0;
   color: #333366;
}
```

Podemos aplicar el mismo estilo en un conjunto de etiquetas. Para ello, indicamos las etiquetas seguidas por comas y luego, entre llaves, los atributos que queramos definir como por ejemplo:

```
h1, p{
   color: red;
}
```

En este caso se define que los encabezados de nivel 1 y los párrafos, tengan letra roja.

3.5.1.2. DEFINICIÓN DE CLASES

Podemos utilizar una clase si deseamos crear un estilo específico, para luego aplicarlo a distintos elementos de la página. Las clases en la declaración de estilos se declaran con un punto antes del nombre de la clase, como por ejemplo:

```
.miclase{
   color: blue;
}
```

Para asignar el estilo definido por una clase en un elemento HTML, simplemente se añade el atributo *class* a la etiqueta que queremos aplicar a dicha clase. El atributo *class* se asigna al nombre de la clase a aplicar. Por ejemplo:

```
<p class="miclase">este párrafo tiene el estilo definido en la clase "miclase".</p>
```

El párrafo anterior se presentaría con color azul.

3.5.1.3. ESTILOS QUE SÓLO SE UTILIZAN UNA VEZ

También podemos tener un estilo específico para un único elemento, que no va a repetirse en ningún otro caso. Para ello tenemos los estilos asignados por identificador. Los identificadores se definen en HTML utilizando el atributo *id* en la etiqueta que deseamos identificar. El valor del atributo *id* será el que definamos nosotros. Por ejemplo:

```
<div id="capa1">
```

En la hoja de estilos, para definir el aspecto de ese elemento con *id* único, se escribe el carácter almohadilla, seguido del identificador indicado en la etiqueta y entre llaves los atributos CSS que deseemos.

```
#capa1{
   font-size: 12pt;
   font-family: arial;
}
```

En este caso se ha asignado fuente de tamaño 12 puntos y cuerpo arial.

Como se puede concluir en la lectura de estas líneas, generalmente se prefiere utilizar estilos definidos en clases a los definidos con identificadores, a no ser que estemos seguros que ese estilo no se va a repetir en todo el documento.

4. ESPECIFICACIÓN DE REQUISITOS

4.1. ALCANCE DEL PROYECTO

Este proyecto abarca la creación, mediante un Servicio Web, de una interfaz programática sobre las aplicaciones ya existentes de dos museos virtuales pertenecientes a la UCM: MIGS y CHASQUI.

Además se desarrolla una aplicación para la gestión de dichos museos que permita comprobar el correcto funcionamiento de las operaciones publicadas por el Servicio Web, así como para ofrecer el acceso a un usuario humano a través de un navegador.

4.1.1. ALCANCE DEL SERVICIO WEB

Ofrece las siguientes operaciones sobre ambos museos:

- Subir un objeto virtual al museo que se ha conectado
- Bajar un objeto virtual del museo que se ha conectado
- Borrar un objeto virtual del museo que se ha conectado
- Listar los objetos virtuales de un museo, tanto si se ha hecho login en dicho museo como si no.
- Buscar objetos virtuales en un museo, tanto si se ha hecho login en dicho museo como si no.

4.1.2. ALCANCE DE LA APLICACIÓN DE GESTIÓN DE MUSEOS

El objetivo de esta aplicación, construida sobre el Servicio Web, es facilitar el acceso a los museos virtuales de MIGS y CHASQUI con fines académicos, demostrando así su funcionamiento. Pretende proporcionar una interfaz intuitiva, abstrayendo la arquitectura subyacente del Servicio Web a los usuarios y facilitando la interoperabilidad.

4.2. REFERENCIAS Y ANTECENDENTES

Se parte de la implementación de las aplicaciones web de los museos como ya se explicó en el apartado *Estado del Arte*.

En un principio la elaboración de este proyecto se apoya en la documentación de un trabajo realizado con anterioridad por Héctor Hernanz '*Gestión distribuida de información en el ámbito de los entornos educativos mediante Servicio Web*' [28].

Dicho trabajo, junto con información recogida tanto de libros como de Internet, ha proporcionado una base sólida de conocimientos sobre Servicios Web en la que se basa este proyecto.

4.3. REQUISITOS DEL SERVICIO WEB

Las operaciones disponibles a través del Servicio Web son:

- **Buscar:**

  Esta operación no requiere conexión. Se podrán buscar objetos virtuales en cualquiera de los dos museos tanto si el usuario posee identificador de usuario y contraseña como si no.

  Recibe como parámetros el nombre del museo en el cual se desea realizar la búsqueda y valores para una serie de atributos, el Servicio Web devuelve la lista de objetos cuyos valores coinciden con los indicados. Para cada objeto se tendrá su identificador, su nombre y una pequeña descripción del mismo.

- **Obtención de atributos:**

  Los atributos sobre los que se realiza la búsqueda dependen de los objetos que se encuentren en cada momento en el repositorio por lo que el Servicio Web ofrece una operación que devuelve estos atributos. Además de esto hay atributos que tienen determinados sus posibles valores, el Servicio Web también ofrece un método que dados un museo y un atributo, devuelve los posibles valores de este último.

- **Listar:**

  Esta operación tampoco requiere conexión. Se podrán listar los objetos virtuales de cualquiera de los dos museos tanto si el usuario posee identificador de usuario y contraseña como si no.

  Debido a la gran cantidad de objetos virtuales que tienen los museos, en especial el CHASQUI, y el tiempo que lleva listarlos todos, se adoptó una solución de petición por lotes, esto es, se solicitan los objetos en pequeños grupos.

De esta forma el Servicio Web devuelve información sobre los diferentes objetos virtuales del lote solicitado, dicha información comprende el identificador de objeto virtual, su nombre y descripción.

- **Bajar:**

  Esta operación requiere conexión al museo, por lo que el usuario ha de proporcionar nombre de usuario y contraseña. Tras conectarse al museo se descarga el objeto virtual solicitado por el usuario a través de su identificador y se procede a la desconexión del mismo.

  Esto se realiza mediante la especificación *SOAP with Attachments*. En este caso se emplea una clase proporcionada por el API de Java, *DataHandler*, para incluir dicho objeto dentro del mensaje SOAP que recibe el cliente del servicio.

- **Subir:**

  Esta operación también requiere conexión al museo. Tras conectarse al museo se procede a insertar en el repositorio del museo el objeto virtual proporcionado por el usuario.

  Al igual que en el apartado anterior se empleará *SOAP with Attachments*, y también a través de la clase *DataHandler*, para incluir el archivo en el mensaje SOAP que recibe el servicio.

  Otra nota importante en esta operación es que los propios museos tienen un límite de objetos a subir, y evidentemente dicho límite tampoco puede ser sobrepasado al subir objetos mediante el Servicio Web.

- **Borrar:**

  Esta operación requiere conexión al museo. Tras conectarse al museo se borra el objeto virtual indicado por el usuario a través de su identificador y se procede a la desconexión del mismo.

  Esta sencilla operación no requiere más que utilizar la función de borrado del museo, comprobando aquellos objetos que no se pueden borrar o bien porque son referenciados por otros objetos o bien porque la implementación interna del museo ha deshabilitado dicha opción. También se ha de tener en cuenta aquellos objetos que no se puedan eliminar porque no se encuentran en el repositorio.

4.4. REQUISITOS DE LA APLICACIÓN DE GESTIÓN DE MUSEOS

Antes de nada, es importante destacar los conceptos que se refieren al cliente del Servicio Web y el usuario de la aplicación final:

> 'Entendemos por *cliente* la aplicación Java que está a un nivel inferior de la interfaz web que es la que ve el *usuario.'*

Más detalladamente, el cliente del Servicio Web, se encarga de conectar con éste para realizar las peticiones solicitadas por el usuario (es decir, es el usuario final), que a través de sus peticiones delega en el cliente a la espera de que éste obtenga una respuesta del Servicio Web para la acción solicitada. La estructura se muestra en la figura 16.

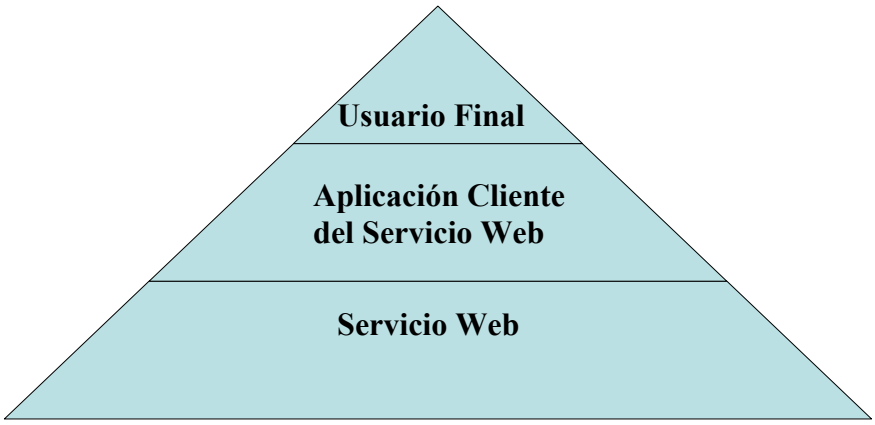

Figura 16. Abstracción de la Estructura de la Aplicación.

4.4.1. FUNCIONAMIENTO, MODO DE PROCEDER Y RESPUESTA

- **Características del Usuario:**

  Aplicación diseñada para dos tipos de usuario:

  - *Investigador*: Tiene acceso a las funciones que manipulan la base de datos, es decir: Subir, Bajar y Borrar un Objeto Virtual del museo *en el que se haya iniciado la sesión.* Es decir, para poder acceder a los Objetos Virtuales de otro museo es necesario hacer login en el correspondiente museo.

  - *Visitantes*: Sólo puede acceder a la búsqueda y listado de los Objetos Virtuales de cualquier museo.

  Tanto investigadores como visitantes, acceden libremente a las funciones de listado y búsqueda sin hacer login.

- **Restricciones:**

  La sesión tiene un tiempo limitado de 30 minutos. Si durante este tiempo no ha habido actividad, ésta se dará por terminada, y será necesario volver a iniciar sesión si se quiere realizar algún tipo de operación.

- **Transcurso de la sesión:**

  Cada tipo de usuario se distingue por un atributo indicando el grupo al que pertenece.

  - *Investigador*: Tras hacer el *login* en el museo solicitado para iniciar la sesión, una serie de atributos se guardan como características de la sesión. Estos atributos permanecen ocultos a lo largo de la sesión y sirven para hacer un seguimiento de esta, de tal modo que el investigador pueda acceder continuamente a las distintas operaciones sin tener que volver a iniciar una sesión nueva.

    Para poder *consultar* las operaciones disponibles, es necesario acceder a la página de 'Acceso Funciones MIGS' o 'Acceso Funciones CHASQUI'. Este link se muestra antes de realizar cualquier operación y en la página de resultados de esta para facilitar así la realización de otra petición.

    La *finalización de la sesión* se lleva a cabo mediante un link que informa del fin de la sesión. En esta página final, se vuelve a dar un acceso a la página principal de la aplicación.

- *Usuario*: en la página de inicio se muestran las funciones disponibles de búsqueda y listado. Estos usuarios también disponen de una sesión que tan sólo guarda un atributo indicando que es un visitante. Sólo le son permitidas las acciones ya mencionadas, que son las de *acceso público*. En el momento que deseen, finalizan la sesión.

4.5. RENDIMIENTO

Es deseable que el tiempo de espera por parte del usuario que quiera acceder al Servicio Web sea el menor posible.

Sin embargo el hecho de implementar la aplicación como un recubrimiento sobre la aplicación web de los museos hace que esta tarea sea complicada, pues obliga a realizar un análisis sintáctico del código HTML generado por dicha aplicación. En casos como la operación listar esto puede ocasionar retardos importantes.

Además de esto, el paso de mensajes mediante SOAP, conlleva un retardo muy pequeño, pero en el caso de emplear SOAP with Attachments se ve incrementado.

4.6. DEPENDENCIA DE LA APLICACIÓN

Se ha intentado desligar lo más posible la implementación del Servicio Web de la de la aplicación web de los museos, pero a pesar de esto, y debido a que el código HTML no tiene unas restricciones fuertemente estrictas, un cambio sustancial en la presentación de los datos de la aplicación web haría que el funcionamiento del Servicio Web se viese seriamente afectado.

4.7. RESTRICCIONES DE DISEÑO

Debido a que partimos de las facilidades de implementación que nos ofrece el proyecto Apache a través de AXIS para la implementación de Servicios Web nos vemos obligados a respetar su estructura de clases.

El código XML se genera de forma transparente al programador pero se debe tener especial cuidado con los tipos de datos que se manejan y en realizar el uso adecuado de la especificación *SOAP with Attachments*.

La implementación de las distintas operaciones se debe adaptar a la implementación de las aplicaciones web existentes.

4.8. ATRIBUTOS DEL SISTEMA SOFTWARE

- *Fiabilidad:* será fiable para garantizar la seguridad de los datos y una realización correcta de la operación solicitada.

- *Disponibilidad:* debe ofrecer una disponibilidad continuada para satisfacer los intereses de los usuarios, pero se ha de tener en cuenta que depende también de la disponibilidad de las aplicaciones originales.

- *Seguridad:* se ajusta a las directivas de privacidad de datos.

- *Mantenibilidad:* su perfecto funcionamiento dependerá de los posibles cambios en las aplicaciones originales por lo que habrá que tener especial cuidado en este aspecto.

- *Portabilidad:* el Servicio Web se puede publicar a través de cualquier equipo que posea el servidor de aplicaciones Tomcat y tenga Axis instalado.

- *Independencia:* el servicio pueda ser ofrecido y utilizado de forma independiente al lenguaje de programación en que se desarrollan las aplicaciones.

- *Interoperabilidad:* Permite que las aplicaciones compartan información y que estando conectado a un museo se pueda acceder a las funciones permitidas del otro.

- *Omnipresencia*: Cualquier dispositivo que trabaje con HTML y XML puede tener acceso a las operaciones ofrecidas por el Servicio Web.

5. DISEÑO DEL SISTEMA

5.1. SERVICIO WEB

El diseño del Servicio Web se muestra en la figura 17.

Figura17. Diseño del Servicio Web

A continuación se detalla la implementación de las operaciones ofrecidas por el Servicio Web:

- *conexionBuscar*: Método público que dado unos requisitos, obtiene los datos de los objetos virtuales que cumplen dichos requisitos.

  - Descripción de sus parámetros de entrada:
    - museo - *String* que indica el museo en el cual deseamos buscar objetos virtuales.
    - entrada - *String[]* con los requisitos que han de tener los objetos virtuales que buscamos.

  - Objeto devuelto:
    - *HashMap* con el identificador de los objetos encontrado y su descripción.

- *conexionListar:* Método público que lista los objetos virtuales del museo desde el correspondiente a fichaInicial hasta el correspondiente a fichaFinal.

  - Descripción de sus parámetros de entrada:
    - museo - *String* que indica el museo del cual deseamos listar un grupo de objetos virtuales.
    - fichaInicial - *String* que indica cuál es la primera ficha cuyo objeto virtual debemos listar.
    - fichaFinal - *String* que indica cuál es la última ficha cuyo objeto virtual debemos listar.

  - Objeto devuelto:
    - *HashMap* con el identificador de los objetos correspondientes al intervalo indicado y su descripción.

- *conexionBajar:* Método público que dado el identificador de objeto virtual, obtiene dicho objeto virtual y lo devuelve, como ya se mencionó anteriormente, mediante un DataHandler.

  - Descripción de sus parámetros de entrada:
    - museo - *String* que indica el museo del cual deseamos descargar el objeto virtual.
    - usuario – *String* con el identificador de usuario requerida por el museo para realizar esta operación.
    - passw – *String* con la contraseña requerida por el museo para realizar esta operación.
    - idOV – *String* con el identificador del objeto virtual que queremos descargarnos.

  - Objeto devuelto:
    - *DataHandler* que contiene el objeto virtual que hemos solicitado descargar.

- *conexionSubir:* Método público que dado un objeto virtual mediante la clase DataHandler incluye éste en el repositorio del museo.

  - Descripción de sus parámetros de entrada:
    - museo - *String* que indica el museo al cual deseamos subir el objeto virtual.
    - usuario - *String* con el identificador de usuario requerida por el museo para realizar esta operación.
    - passw – *String* con la contraseña requerida por el museo para realizar esta operación.
    - archivo - *DataHandler* que contiene el objeto virtual que queremos incluir en el repositorio.

  - Objeto devuelto:
    - *String[]* que contiene dos String, una para indicar si el resultado de la operación ha sido Éxito o Error, y otra para indicar el tipo de error, en caso de producirse, y la finalización del almacenamiento en la base de datos en caso de éxito.

- *conexionBorrar:* Método público que dado un identificador de objeto virtual elimina este del repositorio del museo.

  - Descripción de sus parámetros de entrada:
    - museo - *String* que indica el museo del cual deseamos eliminar el objeto virtual.
    - usuario - *String* con el identificador de usuario requerida por el museo para realizar esta operación.
    - passw - *String* con la contraseña requerida por el museo para realizar esta operación.
    - idOV - *String* con el identificador del objeto virtual que queremos eliminar.

  - Objeto devuelto:
    - *String[]* que contiene dos String, una para indicar si el resultado de la operación ha sido Exito o Error, y otra para indicar el tipo de error, en caso de producirse, y la correcta eliminación del objeto de la base de datos en caso de éxito.

- *obtenerAtributos:* Método público que obtiene los atributos correspondientes a la operación buscar, es decir, aquellos atributos de los objetos virtuales a partir de los cuales se realiza la búsqueda en el repositorio. Este método se emplea con la operación BUSCAR.

  - Descripción de su parámetro de entrada:
    - museo - *String* que indica el museo del cual deseamos obtener sus atributos de búsqueda.

  - Objeto devuelto:
    - *Object[]* con los atributos buscados.

- *obtenerAtributosSeleccion:* Método público para obtener las opciones que ofrece un determinado atributo de selección. Este método se emplea con la operación BUSCAR.

  - Descripción de sus parámetros de entrada:
    - museo - *String* que indica el museo del cual deseamos obtener las opciones corespondientes a un determinado atributo de búsqueda.
    - AtribSelect - *String* con el atributo del cual deseamos obtener sus diferentes opciones de búsqueda.

  - Objeto devuelto:
    - *Object[]* con las opciones corespondientes al atributoindicado.

- *obtenerNombreFichas:* Método público que obtiene el nombre de las fichas, en que se agrupan los objetos virtuales del museo. Este método se emplea con la operación BUSCAR.

  - Descripción de sus parámetros de entrada:
    - museo - String que indica el museo del cual deseamos obtener las diferentes fichas en que se agrupan sus objetos virtuales.

  - Objeto devuelto:
    - Lista tipo Object con los nombres de dichas fichas.

Los siguientes son algunos métodos que emplea internamente la implementación del Servicio Web pero que no son publicados por éste:

- *ponTilde:* Método privado que transforma la representación del símbolo de las tildes en html a la representación ISO de dicha letra.

  - Descripción de sus parámetros de entrada:
    - pal - *String* con la cadena en html que deseamos transformar.

  - Objeto devuelto
    - *String* con la cadena en formato ISO.

- *ponEne:* Método privado que transforma la representación del símbolo de la ñ en html a la representación ISO de dicha letra.

  - Descripción de sus parámetros de entrada:
    - pal - *String* con la cadena en html que deseamos transformar.

  - Objeto devuelto
    - *String* con la cadena en formato ISO.

- *ponPlus:* Método privado que transforma la representación de un símbolo que aparece entre los atributos del museo (±) en html a la representación ISO de dicho símbolo.

  - Descripción de sus parámetros de entrada:
    - pal - *String* con la cadena en html que deseamos transformar.
  - Objeto devuelto
    - *String* con la cadena en formato ISO.
    -

- *ponEspacio:* Método privado que transforma la representación del espacio en html a la representación ISO de éste.

  - Descripción de sus parámetros de entrada:
    - pal - *String* con la cadena en html que deseamos transformar.
  - Objeto devuelto
    - *String* con la cadena en formato ISO.

- *ponPunto:* Método privado que transforma la representación del punto en html a la representación ISO de éste.

  - Descripción de sus parámetros de entrada:
    - pal - *String* con la cadena en html que deseamos transformar.
  - Objeto devuelto
    - *String* con la cadena en formato ISO.

- *quitaTilde:* Método privado que transforma la representación ISO de la tilde a html.

  - Descripción de sus parámetros de entrada:
    - pal - *String* con la cadena con formato ISO que deseamos transformar en html.
  - Objeto devuelto
    - *String* con la cadena en html.

- *quitaEne:* Método privado que transforma la representación ISO de la eñe a HTML.

  - Descripción de sus parámetros de entrada:
    - pal - *String* con la cadena con formato ISO que deseamos transformar en html.
  - Objeto devuelto
    - *String* con la cadena en html.

- *quitaPlus:* Método privado que transforma la representación ISO del símbolo ± a HTML.

  - Descripción de sus parámetros de entrada:
    - pal - *String* con la cadena con formato ISO que deseamos transformar en html.
  - Objeto devuelto
    - *String* con la cadena en html.

- *quitaEspacio:* Método privado que transforma la representación ISO del espacio a HTML.

  - Descripción de sus parámetros de entrada:
    - pal - *String* con la cadena con formato ISO que deseamos transformar en html.
  - Objeto devuelto
    - *String* con la cadena en html.

- *quitaPunto:* Método privado que transforma la representación ISO del punto a html.

  - Descripción de sus parámetros de entrada:
    - pal - *String* con la cadena con formato ISO que deseamos transformar en html.
  - Objeto devuelto
    - *String* con la cadena en html.

5.2. APLICACIÓN DE GESTIÓN DE MUSEOS

El diseño de la aplicación de gestión de museos se ha estructurado siguiendo una variante del modelo *Vista Controlador (MVC)*. Cada página 'sabe' cuál es el manejador más específico que debe tratar la información que se envía en cada petición y le envía al controlador (representado por la clase *Controlador*) quién es este manejador. Después el controlador delega en un manejador el tratamiento de la información. El diseño se puede ver en la figura 18.

- **Clase Controlador**

  Se encarga de iniciar la aplicación. En esta clase se preparan las instancias necesarias para poder realizar todas las peticiones del servidor.

  Para cada tipo de evento que ocurra a lo largo de la aplicación, se define el *manejador* más específico que lo pueda atender. Estos manejadores son:

  - ManejadorLogin
  - ManejadorSube.
  - ManejadorBorra.
  - ManejadorBaja.
  - ManejadorBusca.
  - ManejadorPeticiones.

  Cada uno de estos manejadores (comentados más adelante), está representado por una clase que especializada en el manejo de cada tipo de evento.

Métodos Principales:

- *Init:*
  - Instancia las clases encargadas de manejar cada tipo de evento.
- *DoPost:*
  - Comprueba que los eventos que se dan en cada momento son válidos *(método validateEvent)*
  - Obtiene el manejador más específico *(método getEventHandler)*
  - Delega el manejo de la información de la solicitud en la clase que más especializada para dicha petición *(handler.process)*

- **Clase BaseManejadorEventos**

  Clase más general de la que heredan los manejadores más especializados para manejar cada tipo de evento.

  Métodos Principales:

- Process:
  - Método sobrescrito por cada manejador especializado, ya que son estos los que saben cómo ha de continuar el flujo de la sesión.

- **Clase ManejadorLogin**

  Crea una nueva sesión para cada tipo de usuario (*Investigador o Visitante*). Para ambos queda definido uno de estos dos perfiles.

  Para un *investigador* se fijan como atributos de su sesión el *usuario*, la *contraseña* y el *museo* con el que inició la sesión (queda claro que un visitante no puede tener estos atributos y por lo tanto no hace uso de ellos a lo largo de su sesión).

  En la validación del inicio de sesión no se realiza ninguna llamada al Servicio Web. Sin embargo, sí que se crea una instancia del cliente para el acceso a las funciones publicadas.

  Métodos Principales:

  - Process:
    - Procesa la información enviada por el formulario de login. Llama al método válida para comprobar la información.
  - Valida:
    - Realiza una conexión con el museo en el que se quiere iniciar la sesión para que sea el propio sitio web el que compruebe la veracidad de los datos de login.

  Nota:
  Para los siguientes Manejadores, resaltamos que todos llevan a cabo por igual la tarea de delegar en una entidad concreta (página jsp, html o clase) que ha de continuar con el flujo de la aplicación. Digamos que el 'ManejadorX' se encarga de *decir* quién es el encargado de continuar con la aplicación, mientras que es la clase 'Controlador' quién se encarga de llevar a cabo y ejecutar esta delegación.
  Quedando claro esto, a partir de ahora suponemos que cuando decimos que un manejador 'delega' en una clase, es por mediación de la clase 'Controlador'.

- **Clase ManejadorSube**

  Maneja las peticiones de subida de los objetos virtuales en formato zip al sitio web.

  Métodos Principales:

  - Process:
    Recopila la información necesaria del formulario de subida, es decir, la ruta donde se localiza el Objeto a subir. Una vez obtenidos estos datos, el cliente de la sesión realiza la petición al Servicio Web.
    Si la petición se ha realizado con éxito, la aplicación delega la continuación del flujo de la aplicación en la página 'ResultadoSube.jsp', en caso de fallo, va a la página de Error y se adjunta la causa del fallo, que viene dada por el Servicio Web.

- **Clase ManejadorBorra**

  Maneja las peticiones de borrado de un objeto virtual descrito por un identificador.

  Métodos Principales:

  - Process:
    Recoge la información para el borrado de un Objeto Virtual del formulario HTML de bajada. Esta información consiste en el identificador Objeto Virtual que se desea borrar. El cliente de la sesión se encarga de realizar la petición al Servicio Web.
    Si la petición se ha realizado con éxito, la aplicación delega la continuación del flujo de la aplicación en la página 'ResultadoBorra.jsp', en caso de fallo, va a la página de Error y se adjunta la causa del fallo, que viene dada por el Servicio Web.

- **Clase ManejadorBaja**

  Maneja las peticiones de bajada de un objeto virtual descrito por un identificador.

  Métodos Principales:

  - Process:
    Baja el Objeto Virtual solicitado a través del formulario HTML. Esta petición, igual que las anteriores, la realiza el cliente de la sesión.
    Si la petición se ha realizado con éxito el objeto solicitado se descarga en un directorio del cliente y a continuación se abre en el navegador para permitir al usuario descargárselo. En este caso el flujo de la aplicación se delega en la página 'ResultadoBaja.jsp', pero en caso de fallo, va a la página de Error y se adjunta la causa del fallo, que viene dada por el Servicio Web.

- **ManejadorPeticiones**

  Se encarga de manejar las peticiones públicas que se pueden realizar: listar y buscar.

  Métodos Principales:

  - Process:
    Con la información enviada a través de la petición, sabe discernir el museo en el que se quiere realizar la búsqueda. Además, si el tipo de usuario es un visitante, se crea una sesión para él e inicializa el contador para el listado de los objetos virtuales.

- **Clase ManejadorBusca**

  Maneja las peticiones de búsqueda de un objeto virtual descrito por una serie de atributos cuyos valores han sido descritos en el formulario de búsqueda.

  Métodos Principales:

  - Process:
    Una vez rellenado el formulario 'Busca.jsp', el usuario de la sesión (visitante o investigador) procede a mediar con el Servicio Web para llevarla a cabo. El resultado devuelto consiste en una estructura 'HashMap' en la que como campo clave está el identificador del Objeto Virtual y como información asociada a esta clave, su descripción.
    En caso de éxito, se delega en 'ResultadoBusca.jsp'. En cualquier otro caso se redirecciona a la página de 'Error.jsp'

- **Listar.jsp**

  El listado de los objetos virtuales se genera dinámicamente.
  Recibe un HashMap de la aplicación de gestión (formado por el identificador del OV y su descripción) y presenta la información en lotes de 10 en una tabla. Dependiendo del tipo de usuario, ofrece acceso a las funciones de investigador o va al inicio para poder volver acceder a las públicas.
  La primera vez que se quiere listar, la petición pasa previamente por *ManejadorPeticiones.java* para inicializar el contador.

- **Buscar.jsp:**

  Al igual que el Listar, también se genera dinámicamente, con la diferencia de que aquí *se pretende crear el formulario de búsqueda*, que depende del museo en el que queramos realizar la búsqueda.
  Realiza una llamada al Servicio Web y obtiene como resultado un Array (de Objects) por los que se realizará la búsqueda. Además, para algunos de estos campos es necesario que sus valores pertenezcan a un rango determinado. Por lo que es necesario que el Servicio Web también proporcione una lista con el dominio de cada atributo.

- **ResultadoBusca.jsp:**

  Se genera dinámicamente en función de la lista de objetos encontrados por el Web Service que cumplan los requisitos de búsqueda. Esta lista en realidad es un HashMap, en el que cada campo está formado por el identificador del objeto virtual y su descripción.

- **ResultadoSube.jsp:**

  Página generada en caso de éxito en la operación de subida de un objeto virtual. Informa sobre el resultado de la operación.

- **ResultadoBaja.jsp:**

  Página generada en caso de éxito en la operación de bajada de un objeto virtual. Proporciona información del resultado de bajada. Además emerge una ventana flotante para poder realizar la descarga del objeto virtual en un directorio del usuario.

- **ResultadoBorra.jsp:**

  Página generada en caso de éxito en la operación de borrado de un objeto virtual. Proporciona información sobre el identificador del objeto virtual que se ha borrado.

- **Error.jsp:**

  Página generada en caso de error en cualquier operación, incluyendo login.

- **Clase Gestor**

  Como su propio nombre indica gestiona la concurrencia en la operación de 'Bajada'. Este sistema de concurrencia se basa en la existencia de 10 directorios. El gestor asignará un directorio a cada operación de bajada siempre y cuando éste se encuentre libre. En caso de no haber ningún directorio libre el cliente deberá esperar a que se liberé alguno. Este gestor también se encarga de la liberación de los directorios tras haber sido eliminado su contenido mediante la clase DelFichero.

- **Clase DelFichero**

  Hilo que pasado un determinado tiempo tras la generación de un archivo en uno de los directorios anteriormente mencionados, lo elimina e invoca al gestor para que libere dicho directorio.

  Nota:
  No habrá ningún problema en eliminar un archivo (con el correspondiente *objeto virtual*) pasado un tiempo, pues si la operación se ha realizado con éxito, al abrir el archivo con el navegador éste crea una copia temporal del mismo.

- **Clase Debug**

  Clase estática que sirve para facilitar la información de depuración durante el seguimiento de la sesión

  Métodos Principales:
  - Print, printl.

- **Constants**

  Constantes de las que se hacen uso a lo largo de la aplicación.

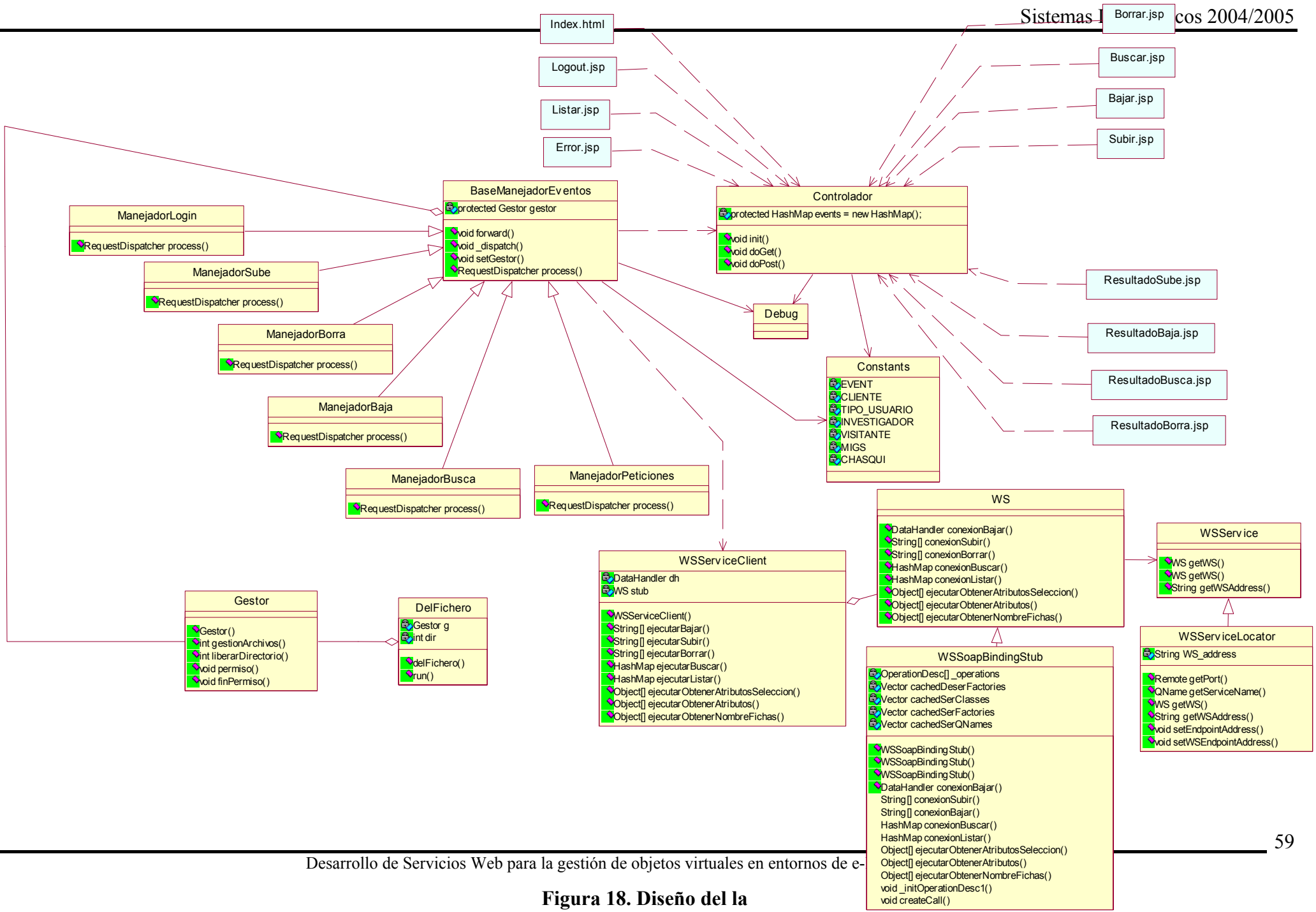

Figura 18. Diseño del la Aplicación de Gestión.

6. MANUAL DE USUARIO DE LA HERRAMIENTA DE GESTIÓN DE MUSEOS

6.1. OBJETIVO

El objetivo de este apartado es introducir a cualquier usuario el modo de proceder a la exploración de la interfaz con el Servicio Web.

6.2. OPERACIONES

6.2.1. INICIO Y LOGIN

Para la parte de usuarios que no tengan acceso al sistema (es decir, no tienen clave de acceso ni nombre de usuario), en ambos lados de la ventana se ofrecen las operaciones públicas de cada museo (buscar y listar). A la izquierda se encuentran las de MIGS y a la derecha las de CHASQUI como se muestra en la figura 19.

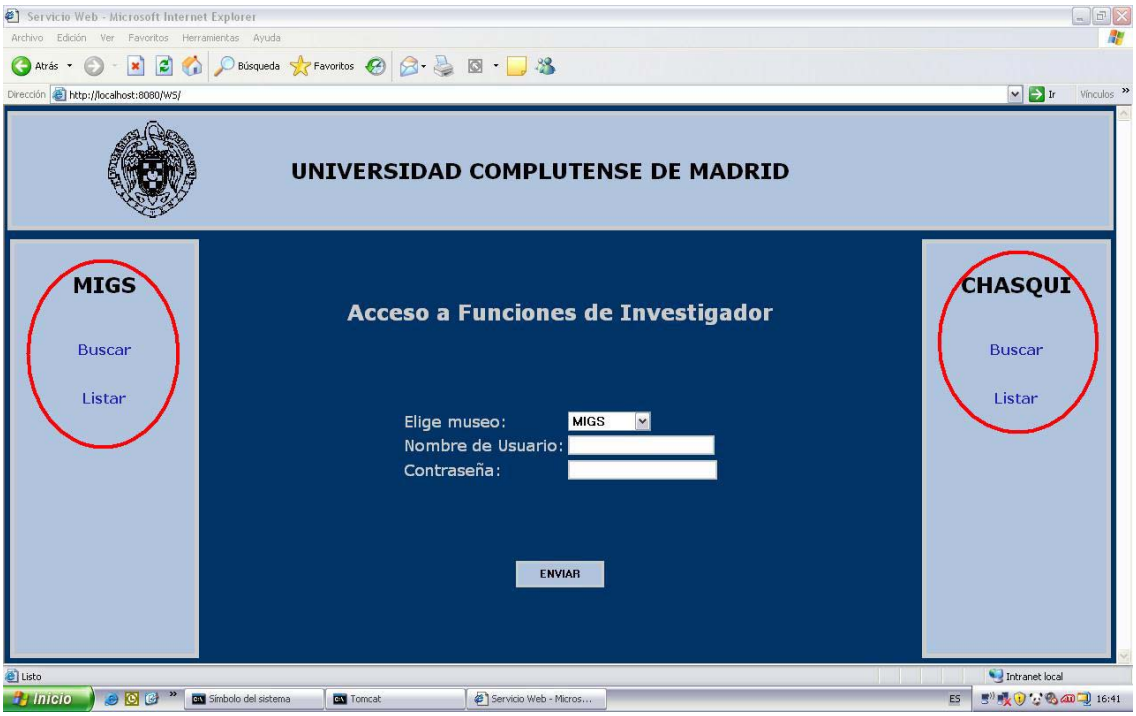

Figura 19. Opciones públicas de los museos.

El resto de usuarios (investigadores) tienen en la parte central los campos para indicar su nombre de usuario y su contraseña, así como un campo para elegir el museo al cual conectarse como se muestra en la figura 20.

Figura 20. Acceso privado a los museos.

A continuación, se explica el modo de proceder para cada función (indistintamente del museo en el que se haga el listado/búsqueda),

6.2.2. OPERACIONES PUBLICADAS POR EL SERVICO WEB

6.2.2.1. OPERACIONES DE ACCESO PÚBLICO

Estas operaciones también se encuentran disponibles para el acceso restringido.

- *Listar:* Hacer click en el vínculo Listar del museo deseado. A continuación, aparecerá la siguiente pantalla, en la que aparecen los 10 primeros objetos virtuales de un museo, correspondientes a las 10 primeras fichas del museo (no a los identificadores de los objetos virtuales almacenados). Se muestra una captura de pantalla en la figura 21.

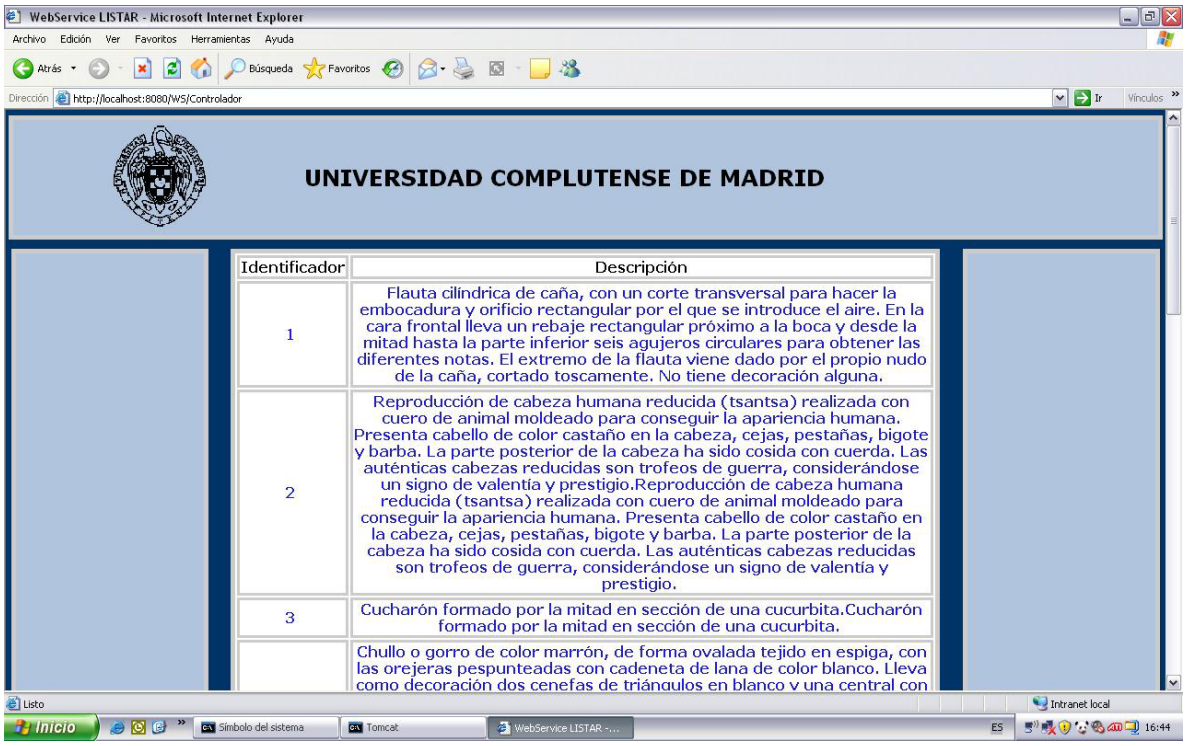

Figura 21. Listado de objetos en el museo CHASQUI.

Para acceder al resto de los objetos (incrementalmente, de diez en diez), hacer click en el vínculo 'Siguiente'.
Para volver al listado de las funciones disponibles, hacer click en 'Acceso Funciones MIGS' o 'Acceso Funciones CHASQUI'(dependiendo del museo en el que nos encontremos haciendo el listado) si estamos registrados y en inicio si no lo estamos.

- *Buscar:* Hacer click en el vínculo Buscar del museo solicitado. A continuación aparecerá una ficha con los datos necesarios que hay que rellenar para realizar la búsqueda. La apariencia de esta ficha, será distinta, dependiendo del museo en el que nos encontremos (MIGS o CHASQUI). Vemos un ejemplo de una plantilla en la figura 22.

Figura 22. Plantilla de búsqueda en el museo CHASQUI.

Una vez rellenados los datos que se hayan estimado oportunos, hacer click en el botón 'Buscar'. A continuación, aparecerá un listado con los objetos encontrados que cumplen las características de la ficha rellenada. Se muestra un ejemplo de resultado en la figura 23.

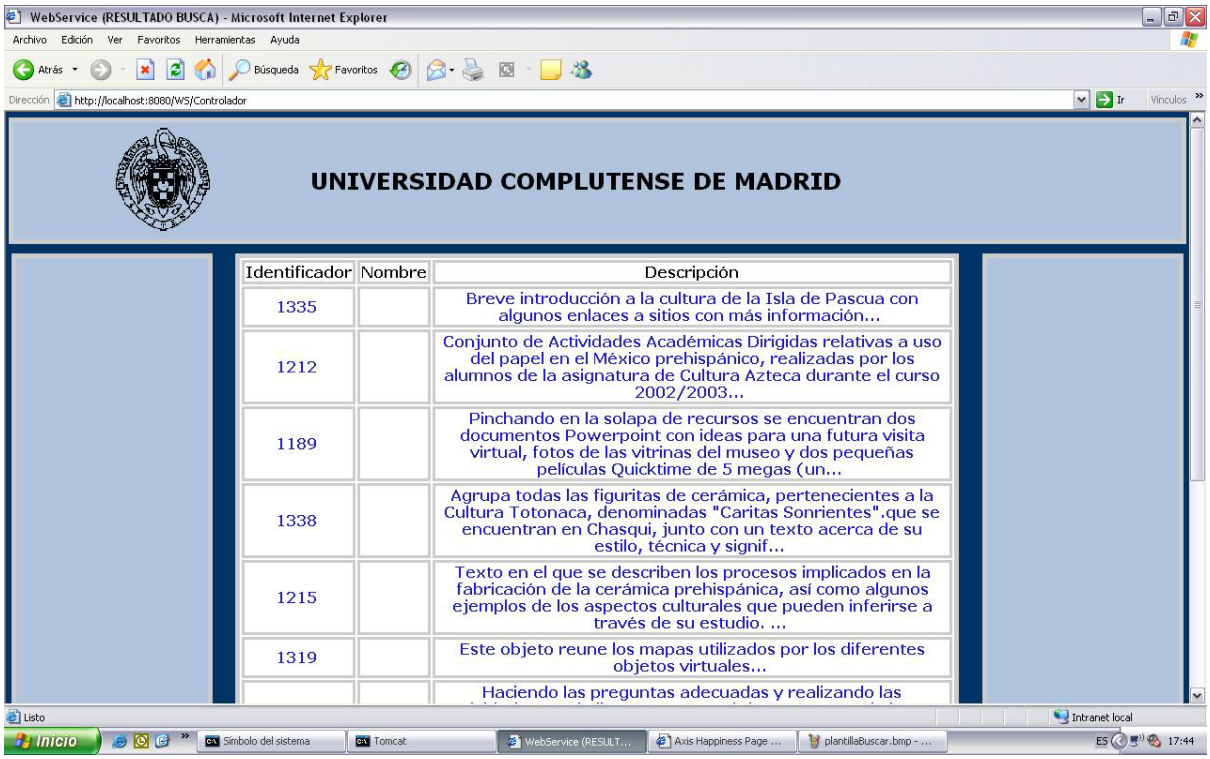

Figura 23. Resultado de búsqueda en el museo CHASQUI.

Para volver a las funciones disponibles, hacer click en 'Acceso Funciones MIGS' o 'Acceso'Funciones CHASQUI' (dependiendo del museo en el que nos encontremos haciendo la búsqueda) si estamos registrados y en inicio si no lo estamos.

6.2.2.2. OPERACIONES DE ACCESO RESTRINGIDO

Además de las anteriores, el usuario con acceso restringido tiene acceso a las siguientes funciones que le permiten manipular los objetos virtuales.

- *Subir Objeto Virtual:* Los Objetos Virtuales han de cumplir la normativa de empaquetamiento. Para que el objeto sea válido, el zip debe tener un archivo manifiesto (imsmanifest.xml), un recurso llamado objetovirtual.xml donde se recogen los datos de las fichas y las dependencias entre ese objeto y otros del museo. En caso de que el objeto empaquetado no sea válido, la aplicación nos lleva a una pantalla de error. Se muestra la plantilla para subir en la figura 24.

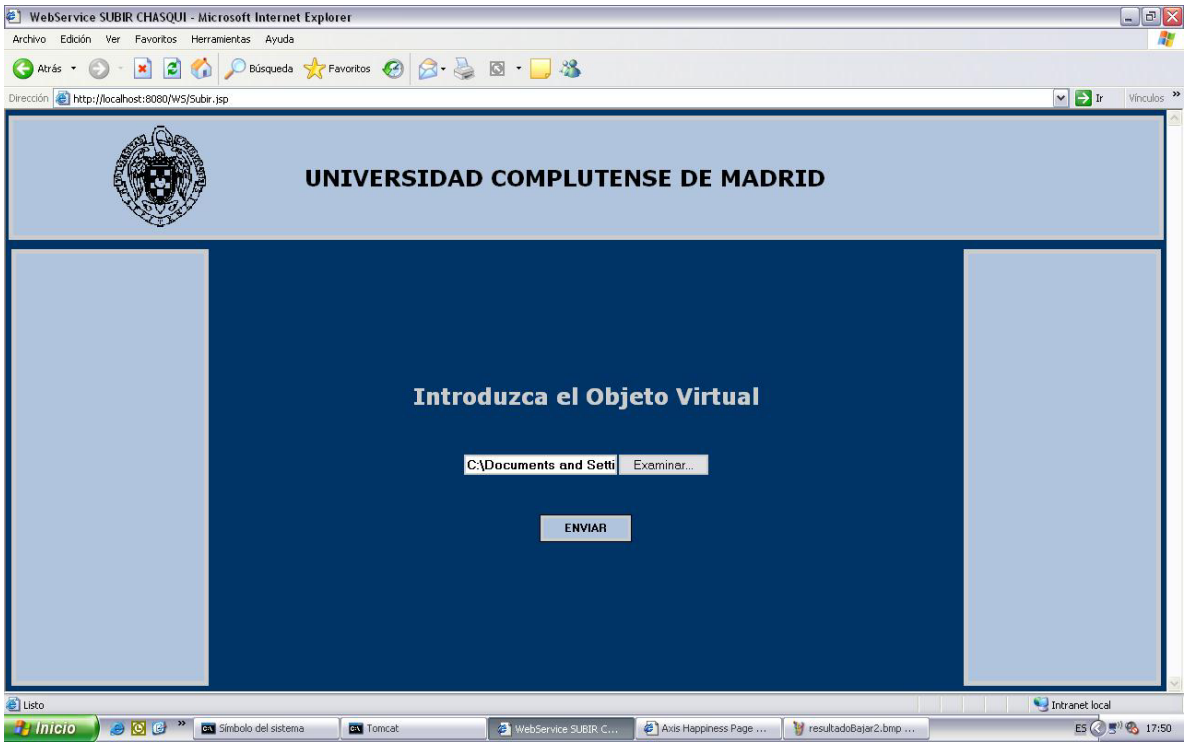

Figura 24. Plantilla para subir objetos virtuales.

Si la subida se ha realizado con éxito, deberá de aparecer la pantalla de la figura 25..

Figura 25. Subida de objeto virtual con éxito.

En ambos casos, se dispone de un vínculo para volver a acceder a las funciones disponibles para cada museo.

- *Bajar Objeto Virtual:* Permite bajar un objeto virtual del museo para que pueda ser utilizado por el usuario. Este objeto baja empaquetado según el estándar de IMS [29]. El objeto que se baja es un archivo zip que contiene los siguientes archivos:

  - El archivo manifiesto (imsmanifest.xml).

  - Los esquemas para validar este archivo xml (ims_xml.xsd, imsmd_v1p2p2.xsd e imscp_v1p1p3).

  - Los recursos propios del objeto, entre los que se encuentra uno especial, objetovirtual.xml. El recurso objetovirtual.xml contiene todos los datos de todas las fichas del objeto.

  - La DTD del objetovirtual.xml.

  - Una hoja de estilo xsl que permite visualizar el archivo objetovirtual.xml, y que se llama ObjetoVirtual.xsl. Abriendo este archivo se tiene una visión del objeto en formato HTML. Este HTML contiene todos los datos de todas las fichas del objeto, así como enlaces a los recursos propios y a otros archivos objetovirtual.xsl de otros objetos virtuales de los que dependía el objeto principal.

  - Todos los objetos de los que depende el objeto virtual también empaquetados, y todos los objetos de los que dependen éstos también empaquetados y en un mismo nivel.

    La pagina para bajar objetos se muestra en la figura 26.

Figura 26. Plantilla para bajar objetos virtuales.

En caso de que hubiese algún problema con esta operación la aplicación redirecciona a una pantalla de error, en caso contrario se abrirá una ventana de descarga en el explorador que permitirá al usuario bajarse el objeto virtual. En caso de éxito se muestra la pantalla de la figura 27.

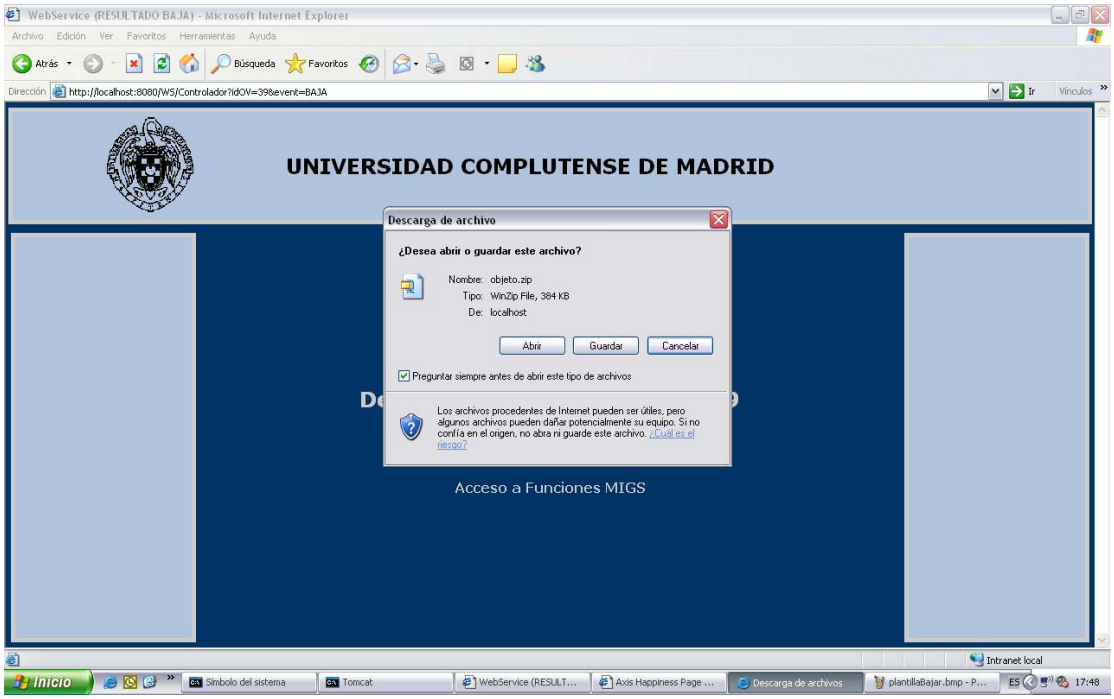

Figura 27. Pantalla de bajada de objeto virtual con éxito.

- *Borrar Objeto Virtual:* Permite borrar un objeto virtual del museo pero sólo se pueden borrar objetos de los que no depende ningún otro objeto, con esto se evitan los posibles errores en la navegación entre objetos relacionados. Para borrar objetos virtuales se nos muestra la pantalla de la figura 28.

Figura 28. Plantilla para borrar objetos virtuales.

En caso de éxito se muestra la pantalla de la figura 29.

Figura 29. Pantalla de borrado satisfactorio de un objeto virtual.

7. CONCLUSIONES Y TRABAJO FUTURO

7.1. CONCLUSIONES

Después de realizar el Servicio Web de acceso a los museos virtuales MIGS y CHASQUI como un recubrimiento de la interfaz web hemos encontrado ciertas ventajas e inconvenientes a este enfoque:

7.1.1. VENTAJAS

- Para realizar el acceso a los objetos virtuales del museo no es necesario conocer su naturaleza, ni cómo están implementados, ni cómo se almacenan en la base de datos. Esto surge como consecuencia de que el acceso no se realiza directamente al repositorio sino a través de la interfaz web, la cual oculta todos estos detalles. Esto permite un desarrollo más rápido, además de facilitar el desarrollo para distintos repositorios con interfaces web similares al no tener que conocer las características particulares de cada modelo de datos.

- La secuencia de pasos es la misma que si nosotros realizáramos el proceso por medio de un navegador. Simplemente se debe simular esta secuencia en un lenguaje de programación y realizar el análisis de la página HTML de respuesta para encontrar la información concreta que necesitamos.

- Al ser los accesos mediante HTTP se pueden realizar desde una máquina remota y no necesariamente en el ordenador que contiene el repositorio. Según esto se podría llegar a la situación mostrada en la figura 30 según la cuál se podría tener en una máquina el repositorio, en otra el Servicio Web y cada cliente en una máquina distinta. Además, en el caso de los dispositivos móviles se podría añadir otra máquina que adaptase las peticiones de estos dispositivos y la respuesta resultante. De este modo tendríamos una arquitectura con un alto grado de distribución.

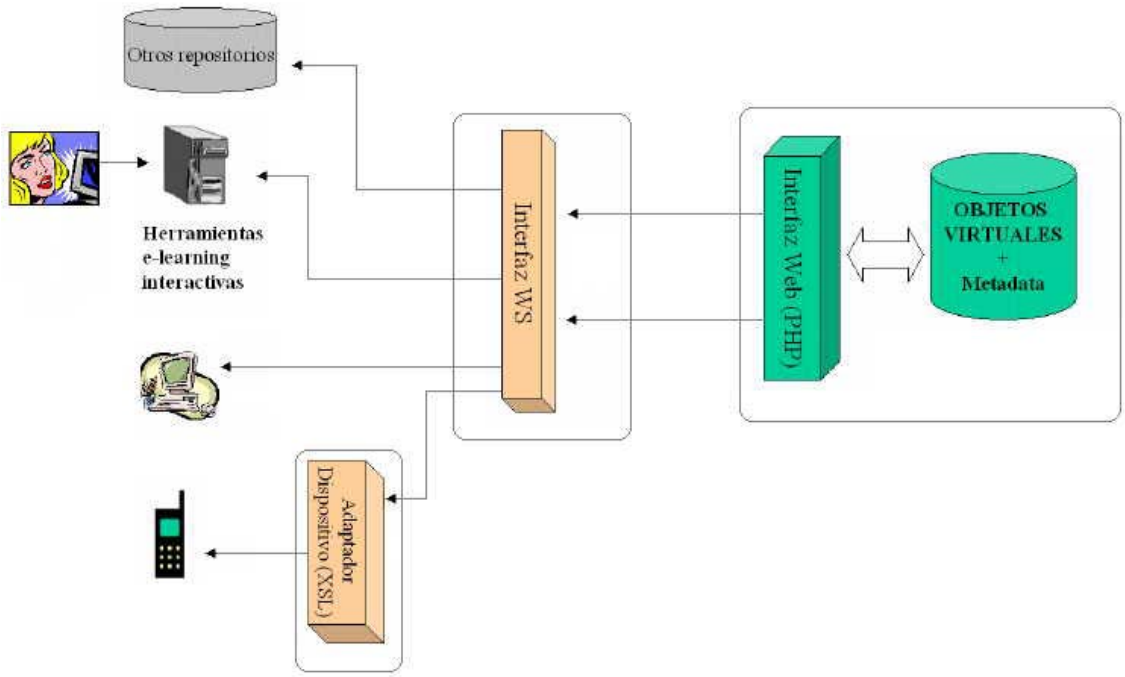

Figura 30. Arquitectura distribuida del Servicio Web y sus clientes.

- Como consecuencia de lo anterior, una persona que no tenga relación con la entidad poseedora del repositorio puede realizar un Servicio Web remoto. Esta persona mediante el uso y estudio de la interfaz web puede desarrollar una aplicación que acceda al repositorio y luego integrarla en un Servicio Web de forma totalmente independiente de la entidad poseedora del repositorio.

7.1.2. INCONVENIENTES

- Al ser HTML un lenguaje con muchas características presentacionales puede provocar errores en el acceso cuando se cambia la interfaz web. El Servicio Web realiza el acceso usando el mismo nombre de variables y sistema de tipos que usa la interfaz web con los métodos POST y GET. Cualquier cambio tanto en el nombre o el tipo de las variables puede provocar que el interfaz web no procese adecuadamente la petición y devuelva un dato incorrecto o un error. También se plantea el mismo problema si se realiza un cambio en el código HTML de la página de respuesta. Al tener que realizarse un análisis del documento de respuesta, un cambio en el código puede provocar que se acceda a información errónea o la imposibilidad de acceder a la información al no entender el nuevo formato. Como consecuencia, cada cambio en la interfaz web provoca que se tenga que probar si el funcionamiento del Servicio Web es correcto.

- Como experiencia personal, se ha comprobado que se deben tratar todos los posibles casos tanto para las peticiones como para las respuestas. La interfaz web en algunas ocasiones devuelve información de las mismas operaciones con

pequeñas diferencias que, si no son tenidas en cuenta a la hora de realizar el análisis de la respuesta, pueden provocar fallos del Servicio Web en determinadas ocasiones. Esto provoca que el desarrollo sea más lento al tener que estudiar todas las posibilidades, así como comprobar que el resultado es el correcto para todas las posibles combinaciones que se puedan dar.

- Como respuesta se obtiene el código HTML que devuelve el interfaz web. Puede que de este código sólo nos interese una pequeña parte, sin embargo debemos esperar a tener toda la página HTML para poder empezar el procesamiento. Esta espera se puede alargar si la respuesta contiene imágenes o gran cantidad de datos que retrasan su transmisión. Esta situación sólo sería problemática en la arquitectura de la figura 16 ya que el tiempo de recepción de la respuesta depende mucho de la conexión entre el Servicio Web y la interfaz web. En el caso de conexiones con poco ancho de banda la espera podría ser muy alta en algunas situaciones. En algunos casos (como listar todos los objetos virtuales del CHASQUI), se ha probado que incluso en conexiones de alta velocidad puede sufrir un alto retardo la respuesta por contener información innecesaria para el Servicio Web pero que penaliza el tiempo de respuesta.

Tras esto se llega a la conclusión de que este tipo de enfoque es muy útil para cuando se quiere hacer un rápido desarrollo de un Servicio Web para probar su utilidad en el repositorio concreto o para definir el interfaz que luego implementará una versión más completa.

También puede ser útil cuando se desarrolle para un sistema que tenga un interfaz web completamente desarrollado que no vaya a recibir muchas modificaciones a lo largo del tiempo que provoquen los problemas anteriormente descritos. Además es aconsejable que los desarrolladores de la interfaz web y del Servicio Web estén en contacto en estas situaciones, sino son la misma persona, para tener inhabilitado el Servicio Web el menor tiempo posible.

7.2. TRABAJO FUTURO

Inicialmente se ha implementado el Servicio Web con acceso directo a la interfaz web del repositorio de CHASQUI y MIGS. En la medida de lo posible se ha hecho este enlace lo menos dependiente posible de la interfaz de presentación, pero en cualquier caso, una modificación sustancial de dicha interfaz afectaría al Servicio Web.

Dentro de la evolución futura se plantea la implementación de las funcionalidades ofrecidas por el Servicio Web, de manera que se realice un acceso directo a la base de datos, y en caso de producirse una modificación en la presentación, el servicio no se vería afectado. Precisamente, una de las ventajas de los Servicios Web es que es perfectamente posible implementar esta evolución a la implementación interna del servicio sin modificar su interfaz, por lo que para las aplicaciones que utilizaran actualmente la interfaz de Servicios Web no deberían realizar ningún cambio en su código.

La arquitectura final sería la mostrada en la figura 2.

Esta arquitectura solucionaría todos los problemas que se han descrito en la sección conclusiones:

- El problema del uso presentacional de HTML se soluciona al eliminarse ese tipo de acceso mediante la interfaz web para obtener la información deseada del repositorio. El acceso se realiza directamente. Con lo cuál se tiene una independencia total de la interfaz pero no pueden adaptarse contenidos.

- Todas las posibilidades en los datos son tratadas por las consultas SQL que se realizarían al repositorio. Por tanto no habría que preocuparse de tantas soluciones distintas. En el caso de una respuesta vacía el tratamiento sería sencillo y en el caso de una respuesta no vacía se leería e interpretaría cada elemento de la respuesta, pero sin tener que tener en cuenta casos excepcionales.

- La petición se realiza mediante una consulta SQL cuya respuesta será un determinado conjunto de tuplas que cumplan la consulta. Por tanto todo lo que se devuelve es importante y no se penaliza el tiempo con datos innecesarios. De este modo se mejora el rendimiento notablemente respecto a la arquitectura intermedia propuesta en este proyecto. La mejora es obvia ya que en la arquitectura del proyecto se realiza la misma consulta por medio del interfaz web más el tratamiento HTML y en esta versión sólo la consulta.

Se propone como enfoque el usar la utilidad JDBC (Java Database Connectivity) [30] de Java para el acceso a bases de datos. Esta aplicación permite un fácil acceso a los datos devolviendo las tuplas que cumplan cada consulta. Con estas tuplas se realizaría el tratamiento adecuado para devolver los datos deseados en el formato XML del Servicio Web.

La secuencia de pasos que sería utilizada puede ser tomada y adaptada del código PHP del interfaz web. En este código, para cada operación se puede ver la secuencia de pasos utilizada que se compone de consultas, agrupaciones de datos, adición de información, etc.

APÉNDICE A:

PLANIFICACIÓN Y DESARROLLO DEL PROYECTO

El Comienzo: Formación, división de tareas y Modelo de Proceso

Desde un principio, la idea del proceso de desarrollo del Servicio Web se basó en la construcción progresiva de prototipos que fueran aumentando su funcionalidad

Antes de empezar el desarrollo, hubo un periodo de formación en las nuevas tecnologías a utilizar (en un principio Servicios Web y Servlets), lo que vino acompañado del estudio de la arquitectura subyacente a éstas: adquisición de conocimientos de AXIS, modo de operar de TOMCAT y APACHE, funcionamiento del protocolo HTTP).

También se realizó una exhaustiva investigación de la arquitectura y funcionamiento de los museos del proyecto CHASQUI actualmente en funcionamiento (de la Facultad de Historia y de la de Informática).

Para llevar esta investigación a cabo, fue necesario:

- Lograr la instalación local de los museos en los PCs de los miembros del equipo (esta tarea fue muy compleja, ya que no había ningún documento que diese instrucciones de instalación ni ofreciese información sobre las versiones utilizadas)

- El estudio de las herramientas ya utilizadas por estos (PHP, SQL)

Una restricción externa identificada como un riesgo desde el principio y que afectaba seriamente al desarrollo del proyecto, fue la disponibilidad de recursos en lo que respectaba a:

- Software: La *falta de privilegios* para el acceso a TOMCAT y sus servicios, en los laboratorios y la instalación en los laboratorios de una *versión no adecuada* para su funcionamiento.

- Hardware: *Insuficiente cuota* en disco para trabajar.

Respecto al *Software*, se desarrolló un archivo de arranque para los componentes necesarios para el funcionamiento (Tomcat5 y JDK1.5). Además, dificultades para conseguir hacerlo funcionar fueron las diferentes configuraciones de las variables de entorno y rutas de cada PC utilizado.

En lo que respecta al *Hardware*, la ayuda del Director del proyecto y la de los técnicos, permitió el aumento de la cuota disponible en algunos de los laboratorios de la facultad.

Se ha realizado un *reparto de tareas* de tal modo que los miembros del equipo pudieran centrarse en cada una de las distintas áreas de desarrollo en las que se ha considerado que se podía dividir el proyecto.

Sin embargo, además de las áreas de conocimiento consideradas inicialmente, ha sido necesario el aprendizaje en nuevas tecnologías (modificación de las bibliotecas de Tomcat, JavaScript, CSS, JSPs) y una mayor indagación en las tecnologías de formación iniciales (Soap with Attachments, HTML)

Es importante resaltar que estas líneas inicialmente divisorias en la formación de los miembros, se han visto alteradas a lo largo del desarrollo del proyecto: los diferentes problemas que han surgido y la solución de estos han requerido la intervención y ayuda de todos. Esto ha sido considerado muy positivo en la formación del grupo, pues todos los miembros han adquirido conocimientos de todos los aspectos de involucrados en la implementación del proyecto.

El desarrollo del Sistema

Una vez formados, en las sucesivas reuniones del equipo, se ha procedido a identificar los requisitos del sistema, la funcionalidad que debía reunir y el modo de proceder de las funciones. Así, los hitos en el desarrollo de nuestro proyecto han sido cada una de las funciones que componen nuestro sistema y la integración de cada una de ellas en la aplicación Web.

El desarrollo del sistema basado en prototipos [31], ha sido una gran ventaja para refinar e identificar los requisitos incrementalmente. Además, ya que los prototipos se han desarrollado en torno a la funcionalidad del sistema, no ha sido necesario desechar los prototipos iniciales (tan sólo modificar algunos detalles en el modo de proceder). El esquema de cómo funciona este modelo se puede ver en la figura 31.

También hemos podido probar distintas alternativas y ver los resultados rápidamente.

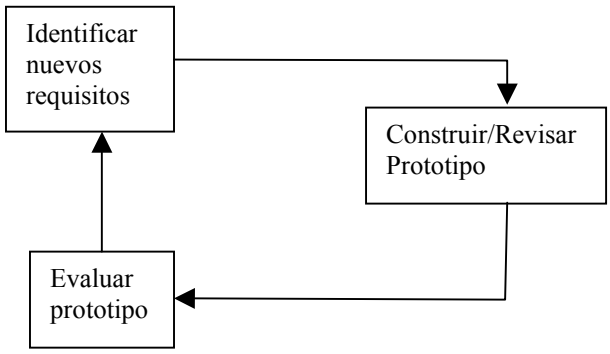

Figura 31. Esquema del modelo de proceso basado en prototipos

Etapa Final

La etapa final y definitiva, ha sido el refinamiento del diseño y documentación del código, así como dotar a la aplicación de una apariencia más amigable e intentar hacer el uso de esta más funcional e intuitivo.

APÉNDICE B:

INSTALACIÓN DE AXIS Y EL SERVICIO WEB

1. Después de descomprimir la carpeta que contiene Axis en una ubicación temporal, debemos realizar su implantación como una aplicación web en el servidor de aplicaciones Tomcat. Para ello copiaremos el directorio axis que se encuentra en la carpeta webapps del directorio temporal en el directorio webapps de Tomcat.

2. Para comprobar que se ha instalado correctamente se accede a http://localhost:8080/axis/

3. Para ver si necesitamos instalar algún componente extra se accede a http://localhost:8080/axis/happyaxis.jsp

4. Las clases correspondientes al Servicio Web se incluirán en axis/WEB-INF/classes

INSTALACIÓN DE LA APLICACIÓN DE GESTIÓN DEL SERVICIO WEB

Al ser una herramienta web lo único que debemos hacer es incluir en la carpeta webapps de Tomcat la carpeta *WS* correspondiente a la aplicación web.

APENDICE C:

GENERACIÓN AUTOMÁTICA DEL FICHEROS DEL SERVICIO WEB MEDIANTE AXIS PASO A PASO

1. Creamos la interfaz de nuestro servicio.

2. El archivo .class obtenido se utiliza para crear el descriptor de servicios, archivo .wsdl. Para ello empleamos la instrucción:

   java org.apache.axis.wsdl.Java2WSDL -o *nombre_del_descriptor*.wsdl -l"http://localhost:8080/axis/services/*nombre_del_servicio*" -n "urn:*espacio_de_nombres*" -p"*paquete*" "urn: *espacio_de_nombres* " *clase*

   • -o para indicar el nombre del archivo WSDL
   • -l indica la localización del servicio
   • -n indica el *namespace* del archivo WSDL
   • -p relaciona el paquete en el que se encuentra con el *namespace* .
   • al final ponemos el nombre específico de la clase que contiene el Servicio Web.
   (Nota: El archivo wsdl se creará en el directorio en el que nos encontremos)

3. Generamos los archivos .java necesarios para el Servicio Web, tanto los del lado del servicio como aquellos que necesita el cliente, mediante la instrucción:

   java org.apache.axis.wsdl.WSDL2Java -o . -d Session -s -S true -Nurn: *espacio_de_nombres clase nombre_del_descriptor*.wsdl

   Para ello tenemos que encontrarnos en el directorio donde se encuentra el archivo *nombre_del_descriptor*.wsdl.

4. Incluimos la implementación específica para las operaciones de nuestro servicio en el archivo *SOAPBindingImpl.*

5. Compilamos los .java de la *interfaz*, el archivo *skeleton* y *SOAPBindingImpl* e incluimos los .class obtenidos en el directorio webapps\axis\WEB-INF\classes de Tomcat.

6. Ahora ejecutamos java org.apache.axis.client.AdminClient deploy.wsdd para publicar nuestro servicio. El archivo deploy.wsdd también ha sido generado automáticamente por AXIS en el paso anterior.

El cliente se servirá de los archivos *stub* y de *localización* que se han generado para hacer uso del servicio.

BIBLIOGRAFÍA

[1] Fernandez-Valmayor, A., Guinea, M., Jiménez, M., Navarro, A., Sarasa, A.. *Virtual Objects: an Approach to Building Learning Objects in Archeology* en Computers and Education: Toward a Lifelong Learning Society. M. Llamas-Nistal, M.J. Fernández-Iglesias and L.E. Anido-Rifón.(Eds.).12 pp. En prensa. Kluwer Academic Publisher. Dordrecht, The Netherlands

[2] Sierra, J.L., Fernández-Valmayor, A., Guinea, M., Hernanz, H., Navarro, A., *Building Repositories of Learning Objects in Specialized Domains: The Chasqui Approach*

[3] Navarro, A., Sierra, J.L., Fernández-Valmayor, A., *From Chasqui to Chasqui II: an Evolution in the Conceptualization of Virtual Objects*

[4] PHP: www.php.net

[5] ASP: www.asp.net

[6] JSP: http://java.sun.com/products/jsp/

[7] ODBC: http://www.webopedia.com/TERM/O/ODBC.html

[8] MySQL: www.mysql.com

[9] GPL: http://www.gnu.org/copyleft/gpl.html

[10] Hernanz, H., Fernández-Valmayor, A. *Web Service and Virtual Objects in Heterogeneous Educational Environments*

[11] XSL: http://www.w3.org/Style/XSL/

[12] HTTP: http://www.w3.org/Protocols/

[13] FTP: http://www.w3.org/Protocols/rfc959/

[14] XML: www.xml.com

[15] WSDL: http://www.w3.org/TR/wsdl

[16] UDDI: http://www.uddi.org/

[17] SGML: http://www.w3.org/MarkUp/SGML/

[18] HTML: http://www.w3.org/MarkUp/

[19] XML-RPC: http://www.xmlrpc.com/

[20] SOAP: http://www.w3.org/TR/2000/NOTE-SOAP-attachments-20001211

[21] MIME: http://www.mhonarc.org/~ehood/MIME/

[22] XML Schemas: http://www.w3.org/XML/Schema

[23] DTD: http://www.w3schools.com/dtd/default.asp

[24] AXIS: http://ws.apache.org/axis/

[25] JAVA: http://java.sun.com/

[26] TOMCAT: http://jakarta.apache.org/tomcat/

[27] Servlet: http://java.sun.com/docs/books/tutorial/servlets/

[28] Hernanz, H. *Gestión distribuida de información en el ámbito de los entornos educativos mediante Web Services*. Trabajo de tercer ciclo dirigido por Alfredo Fernández-Valmayor

[29] IMS: www.imsglobal.org

[30] JDBC: http://java.sun.com/products/jdbc/

[31] Pressman, R. *Ingeniería del Software. Un enfoque práctico*; 5ª edición, Ed. McGraw-Hill, 2001

GLOSARIO

- **AXIS** Apache Axis es una implementación de SOAP (Apartado3.7)
- **Interoperabilidad** Permite que las aplicaciones compartan información (Apartado 2.3)
- **Java** Lenguaje de programación orientado a objetos. (Apartado 3.1)
- **Museo Virtual** Un museo virtual es un interfaz accesible a través de cualquier computador que tenga acceso a la red y disponga de un navegador web y que permite consultar objetos virtuales. (Apartado 1.1.1)
- **Servicio Web** Es una unidad lógica de aplicación que ofrece datos y servicios a otras aplicaciones (Apartado 2.1)
- **SOAP (Simple Object Access Protocol)** Permite que programas que corren en diferentes sistemas operativos se comuniquen. (Apartado 2.7)
- **UDDI (Universal Description Discovery and Integration)** Permite la localización y publicación de los Servicios Web. (Apartado 2.7)
- **WSDL (Web Services Definition Service)** Se encarga de describir el Servicio Web cuando es publicado. (Apartado 2.7)
- **XML (Extensible Markup Language)** Es un metalenguaje de marcado en el cual se basan los Servicios Web (Apartado 2.7)

Se autoriza a la Universidad Complutense de Madrid a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto en la propia memoria, como el código, la documentación y/o el prototipo desarrollado para la asignatura Sistemas Informáticos del curso 2004/2005 bajo el nombre 'Desarrollo de Servicios Web para la gestión de objetos virtuales en entornos de e-learning'.

Los autores:

Laura Gómez Pérez

Beatriz Molina Sánchez

Álvaro Rodrigo Yuste